UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
INSTITUTO DE INFORMÁTICA
CURSO DE CIÊNCIA DA COMPUTAÇÃO

RODRIGO BRAUWERS

Estendendo e Instanciando o Game Agile Methods Applied (GAMA)

Trabalho de Graduação.

Prof. Dr. Marcelo Soares Pimenta
Orientador

Prof. Me. Fábio dos Santos Petrillo
Co-orientador

Porto Alegre, julho de 2011.

AGRADECIMENTOS

Agradeço aos que fizeram com que este trabalho pudesse ser realizado: Deus, meus pais, meus irmãos e meus orientadores (Fábio Petrillo e Marcelo Pimenta).

SUMÁRIO

LISTA DE ABREVIATURAS E SIGLAS

FPS	*First-person shooter*
GAMA	*Game Agile Methods Applied*
GDD	*Game Design Document*
GUP	*Game Unified Process*
IA	Inteligência Artificial
MA	Modelagem Ágil
PB	*Product Backlog*
RUP	*Rational Unified Process*
UFRGS	Universidade Federal do Rio Grande do Sul
UML	*Unified Modeling Language*
XGD	*Extreme Game Development*
XP	*Extreme Programming*

LISTA DE FIGURAS

LISTA DE TABELAS

RESUMO

A indústria de jogos é atualmente a mais lucrativa do ramo do entretenimento. A complexidade de desenvolver jogos divertidos é proporcionalmente grande. As dificuldades e necessidades inerentes ao universo de desenvolvimento de jogos leva muitos projetos ao fracasso, sendo que a maior parte das restrospectivas sobre projetos de jogos relata algum tipo de agonia, pressão e estresse.

Vários autores e profissionais envolvidos no desenvolvimento de jogos acreditam que as práticas ágeis podem ajudar a minimizar ou eliminar alguns dos problemas comuns nessa área. Nesse contexto, algumas metodologias ágeis específicas para o desenvolvimento de jogos foram criadas, sendo que uma delas é o GAMA – *Game Agile Methods Applied.*

O objetivo deste trabalho é analisar e estender o GAMA para torná-lo mais completo em relação as necessidades envolvidas no desenvolvimento de jogos e aplicar algumas práticas ágeis que o compõe num estudo de caso. Nesse ínterim, serão considerados alguns problemas frequentemente mencionados por desenvolvedores de jogos e como as práticas ágeis podem ser úteis nesse contexto.

Palavras-chave: GAMA, Game Agile Methods Applied, jogos, desenvolvimento de jogos, metodologias, práticas ágeis.

Extending and Instantiating the Game Agile Methods Applied (GAMA)

ABSTRACT

The game industry is currently the most profitable branch of entertainment. The complexity of developing fun games is proportionately large. The difficulties and needs inherent in the universe of game development takes many projects to fail, and most of the game projects on retrospective reports some kind of agony, pressure and stress.

Several authors and professionals involved in game development believe that agile practices can help to minimize or eliminate some of the common problems in this area. In this context, some agile methodologies specific to the development of games were created. One of that is GAMA - Game Agile Methods Applied.

The objective of this study is to analyze and extend the GAMA to make it more complete in relation to the needs involved in game development and apply some of it's agile practices in a case study. Meanwhile, will be considered some problems often mentioned by game developers and how agile practices can be useful in this context.

Keywords: GAMA, Game Agile Methods Applied, games, game development, methodologies, agile practices.

1. INTRODUÇÃO

1.1 Contextualização

Nos primórdios da indústria de jogos, desenvolver um jogo era uma atividade relativamente simples. Um desenvolvedor podia criar um jogo em questão de dias. Não eram necessários uma equipe repleta de *designers*, artistas, programadores de Inteligência Artificial, etc. Bastava ter paixão por jogos e um pouco de conhecimento técnico. Naquele tempo, cada projeto de jogo era orçado em alguns milhares de dólares. Podia-se dar o luxo de testar ideias sem se comprometer freneticamente com resultados. Cada projeto que obtinha sucesso financiava uma dezena de novos projetos [KEITH, 2010].

Muita coisa mudou desde então. A indústria de jogos cresceu a ponto de se tornar o principal ramo da indústria do entretenimento. Seguem abaixo alguns dados nesse respeito:

- Em 2009, a indústria de jogos lucrou US$10,5 bilhões [ESRB, 2011]
- O terceiro homem mais rico do Japão é Hiroshi Yamauchi, principal presidente da história da Nintendo [WIKIPEDIA, 2011]
- O jogo *Call of Duty: Black Ops* vendeu em 5 dias o equivalente a US$650 milhões, um recorde entre *video games*, filmes e livros [WIKIPEDIA, 2011]
- De janeiro a março de 2011, as vendas de jogos cresceram 57% em relação ao mesmo período de 2010 [BUSINESS WEEK, 2011]
- 67% das famílias americanas jogam *video games* [ESRB, 2011]
- 40% dos jogadores são mulheres [ESRB, 2011]
- A idade média dos jogadores é de 34 anos [ESRB, 2011]

Os dados acima dão uma pequena ideia do “monstro” em que se tornou a indústria de jogos. É interessante notar que a prática de jogar *video games*, muitas vezes relacionadas a jovens do sexo masculino, é, na realidade uma prática comum também entre mulheres e adultos [ESRB, 2011].

Assim como o tamanho da indústria de jogos aumentou vertiginosamente nos últimos anos, a complexidade envolvida em desenvolver jogos cresceu proporcionalmente. Daquele desenvolvedor solitário que implementava o jogo do início ao fim em questão de dias, passou-se a equipes multidisciplinares com mais de cem pessoas em alguns casos, sendo que a duração dos projetos de jogos pode chegar a quatro anos [BLOW, 2004; GERSHENFELDT, LOPARCO, BARAJAS, 2003].

1.2 Motivação

O cenário de grande complexidade envolvido no desenvolvimento de jogos parece indicar a necessidade de que boas práticas e metodologias sejam utilizadas. Soma-se a isso o fato de que muitos projetos de jogos tem seus custos orçados em dezenas de milhões de dólares. Nestes casos, o fracasso simplesmente não é uma opção, sendo que boas práticas e metodologias podem aumentar a chance de sucesso [KEITH, 2010].

Surpreendentemente, embora a indústria de jogos como um todo seja bastante lucrativa, os casos de fracassos e falência são muito comuns. Para cada jogo de sucesso, há vários projetos abandonados ou que resultaram em prejuízos [CALLELE, NEUFELD, SCHNEIDER, 2005]. Por exemplo, ao analisar postmortens[1] , dificilmente são encontrados casos nos quais não se passou por algum tipo de problema, pressão ou agonia.

Além do fato de o desenvolvimento de jogos ser naturalmente complexo, parece que muitas equipes sofrem de problemas, especialmente pela falta de boas práticas ou processos e metodologias inadequados [PETRILLO, 2008]. Vários autores indicam que o processo tradicional de desenvolvimento de jogos, o modelo em cascata (*Waterfall*), não supre as necessidades específicas dessa área e é potencialmente prejudicial [FLOOD, 2003; GIBSON, 2007; KEITH, 2010]. Estes mesmos autores defendem que a utilização de práticas ágeis é o melhor caminho a seguir. As pecularied ades da indústria de jogos fazem com que Callele, Neufeld e Schneider [2005] sugiram a necessidade de metodologias de desenvolvimento específicas para esta área.

Refletindo sobre isso, Petrillo [2008], em trabalho publicado em 2008, analisou os principais problemas enfrentados no desenvolvimento de jogos e agrupou um conjunto de práticas ágeis que potencialmente minimizam ou eliminam tais problemas. O resultado final foi o *Game Agile Methods Applied*, ou GAMA, composto por práticas incentivadas pela Modelagem Ágil, *Extreme Programming*, *Scrum* e Desenvolvimento Enxuto de *Software* [PETRILLO, 2008].

A motivação deste trabalho é, essencialmente, analisar e estender o GAMA para torná-lo mais completo em relação as necessidades envolvidas no desenvolvimento de jogos. Nesse ínterim, iremos abordar algumas questões: Porque o processo tradicional de desenvolvimento de jogos não é adequado? O que nos leva a crer que as práticas ágeis produzem resultados mais satisfatórios? Quais aprimoramentos que condizem com as necessidades dos projetos de jogos e com as práticas ágeis podem ser incorporados ao GAMA? Como estender o GAMA para deixá-lo mais completo? Como as práticas ágeis que compõe o GAMA podem ser aplicadas? Quais são os benefícios de aplicá-las?

1.3 Objetivos

O principal objetivo deste trabalho é analisar o GAMA, identificar alguns aprimoramentos e então propor uma versão estendida que contemple melhor as necessidades de projetos de desenvolvimento de jogos.

[1] Documento onde é feita uma retrospectiva do projeto, relatando-se o que deu certo e o que deu errado

Além do objetivo principal do trabalho, temos alguns objetivos específicos:

- Elencar os principais problemas do processo tradicional de desenvolvimento de jogos, conforme encontrado nas referências da área
- Justificar a aplicação de práticas ágeis no desenvolvimento de jogos
- Apresentar a versão original do GAMA
- Analisar o GAMA a luz das práticas ágeis e das necessidades peculiares do desenvolvimento de jogos
- Identificar possíveis aprimoramentos ao GAMA
- Incorporar os aprimoramentos identificados numa versão estendida do GAMA
- Demonstrar parcialmente a aplicação do GAMA utilizando um estudo de caso

1.4 Estrutura do trabalho

Para contemplar os objetivos propostos, o trabalho foi dividido em 5 capítulos. Após a introdução, no capítulo 2 apresentaremos, inicialmente, algumas justificativas para a utilização das práticas ágeis no desenvolvimento de jogos eletrônicos. Vamos abordar o processo tradicional de desenvolvimento de jogos e analisar alguns de seus problemas. Então iremos discorrer sobre algumas características das práticas ágeis que potencialmente as tornam adequadas para o desenvolvimento de jogos. Na sequência, iremos apresentar o estado da arte na forma de quatro metodologias ágeis aplicadas no desenvolvimento de jogos. A última metodologia a ser apresentada, o GAMA, ganhará destaque especial visto ser o principal objeto de estudo deste trabalho.

No capítulo 3 iremos realizar uma análise sobre o GAMA. Esta análise será realizada principalmente a luz das necessidades dos projetos de desenvolvimento de jogos e das práticas ágeis. Como resultado da análise, identificaremos alguns aprimoramentos que podem ser incorporados ao GAMA. Na segunda parte do capítulo, iremos estender o GAMA de modo que os aprimoramentos identificados sejam contemplados e a metodologia se torne mais completa em relação as necessidades desta área de desenvolvimento.

Na sequência do trabalho, no capítulo 4, serão apresentados os resultados obtidos pela aplicação de algumas práticas, valores, atividades e características ágeis que fazem parte do GAMA num estudo de caso. Abordaremos alguns aspectos de tais atividades que foram úteis no desenvolvimento do estudo de caso e, potencialmente, as tornam apropriadas para o desenvolvimento de jogos em geral.

Finalizando o trabalho, iremos apresentar a Conclusão no capítulo 5. Vamos elencar o resumo dos resultados produzidos, as limitações enfrentadas e as perpectivas de trabalhos futuros.

2. AGILIDADE EM PROJETOS DE DESENVOLVIMENTO DE JOGOS

Neste capítulo iremos relacionar as práticas ágeis com o desenvolvimento de jogos. Na seção 2.1, iremos apresentar algumas razões que levam a crer que o uso de práticas ágeis no desenvolvimento de jogos é justificável e necessário. Na seção 2.2, apresentaremos quatro metodologias ou processos ágeis utilizados no desenvolvimento de jogos.

2.1 Práticas ágeis e o desenvolvimento de jogos

Entre os profissionais envolvidos no desenvolvimento de jogos, é senso comum o fato de que desenvolver jogos é uma atividade extremamente complexa [GERSHENFELDT, LOPARCO, BARAJAS, 2003], sendo que esta complexidade tem aumentado significantemente nos últimos anos [BLOW, 2004]. Gibson [2007] relata o exemplo de um jogo que em dez anos teve o número de componentes aumentados de cinco para doze, sendo que cada componente era em muito mais complexo do que originalmente. Além disso, a indústria de jogos enfrenta não apenas os problemas comuns na indústria de *software*, mas também alguns problemas peculiares [PETRILLO, 2008; PETRILLO, PIMENTA, TRINDADE, DIETRICH, 2009]. Por exemplo, pode ser muito desafiador administrar com sucesso a multidisciplinaridade naturalmente envolvida no desenvolvimento de jogos [CALELLE, NEUFELD, SCHNEIDER, 2005].

A dificuldade envolvida em projetos de jogos eletrônicos é tamanha que são encontradas declarações como estas:

> *"É raro um projeto que não tenha algum tipo de agonia. Estou tentando pensar em um projeto que não tenha sido miserável em algum aspecto. Posso pensar em um que tenha sido muito consistente, mas em geral, penso que encontrará sempre algum grau de angústia associada a criação de jogos."* Ian Lan, Mac Doc Software [GERSHENFELD; LOPARCO; BARAJAS, 2003]

> *"Todo o projeto tem um momento no qual você está totalmente no inferno, especialmente quando estiver criando uma nova propriedade intelectual ou tentando criar a "diversão" e a identidade central do jogo."*
> Scott Campbell, Incognito Studio [GERSHENFELD; LOPARCO; BARAJAS, 2003]

Diante deste cenário de agonia e pressão - muitas vezes considerados normais no desenvolvimento de jogos - cabe perguntar: qual é a origem desses problemas? Para

muitos, a resposta, pelo menos em parte, está no processo tradicional de desenvolvimento de jogos.

2.1.1 O processo tradicional de desenvolvimento de jogos

Segundo Flood [2003], a maior parte da indústria de desenvolvimento de jogos utiliza, de alguma forma, o modelo em cascata, ou *Waterfall* [PRESSMAN, 2006], sendo que este pode ser considerado o processo tradicional e padrão nessa área.

Resumidamente, o modelo em cascata, quando aplicado ao desenvolvimento de jogos, faz com que primeiro as regras do jogo sejam criadas, depois documentadas e projetadas, então desenvolvidas e finalmente testadas. Como o nome sugere, o modelo em cascata é linear: a *priori*, as fases concluídas não são revisitadas. Assim, cada fase é executada numa ordem predeterminada e depende totalmente da fase anterior. A figura 2.1 mostra uma instância do modelo em cascata aplicado ao desenvolvimento de jogos.

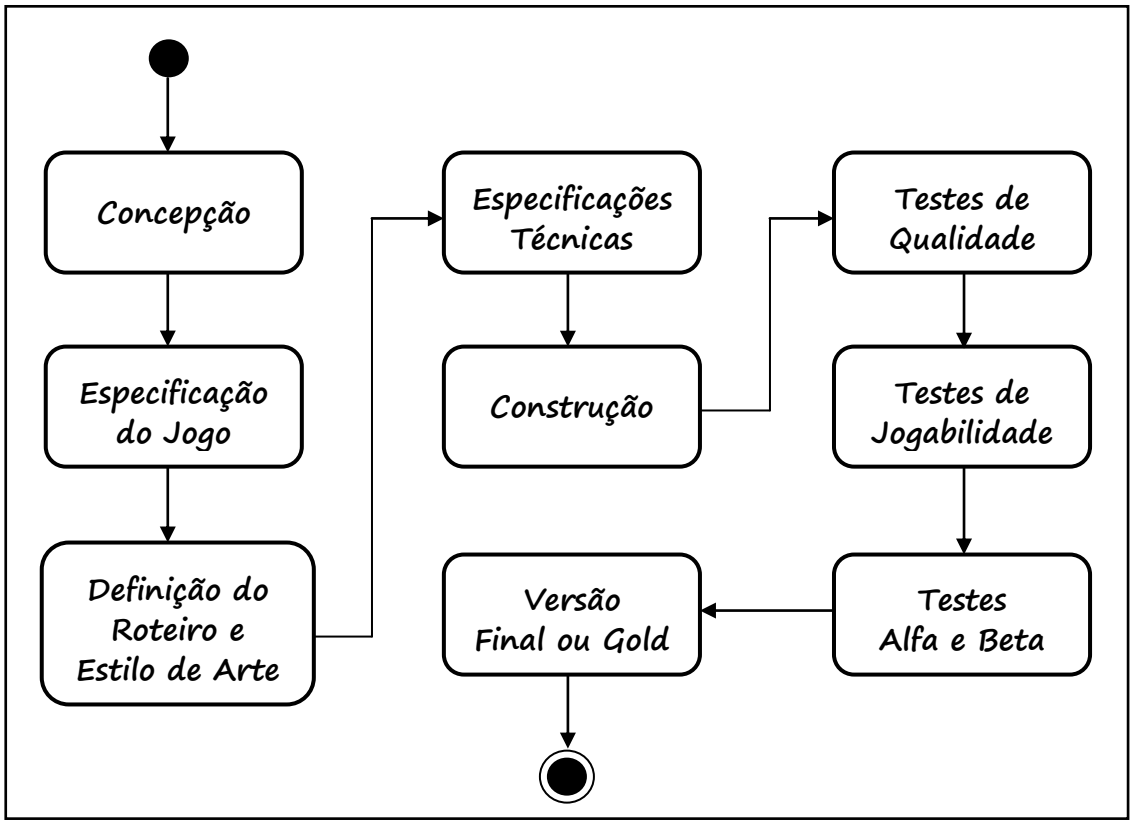


Figura 2.1. Processo em cascata adaptado ao desenvolvimento de jogos [PETRILLO,2008]

2.1.2 Problemas do processo tradicional desenvolvimento de jogos

Embora alguns autores defendam a utilização do modelo em cascata no desenvolvimento de jogos [BETHKE, 2003], muitos outros argumentam que esse apresenta muitos problemas, não é apropriado e é potencialmente prejudicial. Por

exemplo, Flood [2003] argumenta que o processo em cascata não permite que o jogo seja testado enquanto ainda há tempo para mudanças e dificulta a comunicação, visto que apenas alguns grupos de pessoas tem acesso ao jogo em cada etapa.

Gibson [2007] destaca que no modelo em cascata muitas vezes um desenvolvedor precisa esperar que outro finalize suas atividades para só então começar a trabalhar. Por exemplo, um programador de Inteligênica Artificial (IA) pode ficar ocioso esperando a definição completa dos elementos envolvidos. Em contrapartida, quando esta definição lhe é entregue, o programador subitamente se vê confrontado com muito trabalho a fazer e, potencialmente, se sente precocemente pressionado.

McGuire [2006] defende que os testes e integração executados tardiamente no modelo em cascata resultam em perda de tempo, dinheiro e de fé no projeto, além de um produto de pior qualidade. Por exemplo, uma *feature* pode ser amplamente utilizada no projeto do jogo e dos níveis, cujas definições são realizadas nas fases iniciais. Mas e se no fim do projeto, quando o jogo for testado, descobrir-se que essa *feature* não é tão divertida quanto se pensava? Neste caso, há um problema que pode comprometer a viabilidade e o sucesso do jogo.

Outro problema do modelo em cascata identificado por McGuire [2006] é o fato de que, devido a linearidade imposta pelo processo, enquanto alguns desenvolvedores estão muito ocupados e possivelmente apressando seu trabalho (resultando em pior qualidade), outros estão esperando sem nenhuma atividade a realizar. Além disso, segundo McGuire [2006], os *designers* tem seu trabalho dificultado, visto que coisas importantes do jogo, como câmera, personagens e IA são entregues apenas nas fases finais do projeto.

Segundo Keith [2010], no modelo em cascata a diversão do jogo é testada muito tardiamente, quando as alterações nessárias detectadas nos testes são muito difíceis e custosas de desenvolver. Além disso, Keith [2010] menciona que no modelo em cascata o conhecimento e experiência adquiridos durante o desenvolvimento do jogo são muito pouco aproveitados, visto que o planejamento realizado nas primeiras etapas do projeto dificilmente é alterado. Adicionalmente, problemas como o *feature creep*[1] e cronogramas otimistas são potencializados, pois as decisões mais importantes do projeto são tomadas quando as incertezas são maiores.

2.1.3 As práticas ágeis como proposta de solução

Afirmar categoricamente que as práticas ágeis são a solução para os frequentes problemas no desenvolvimento de jogos é precipitado e incorreto. Conforme sugerido pelo Manifesto Ágil [AGILE ALLIANCE, 2006], a questão principal são as pessoas, não os processos ou metodologias. O primeiro valor do Manifesto Ágil é:

"We value individuals and iterations over processes and tools"

[1] Feature creep: tendência de colocar novas funcionalidades no jogo durante o desenvolvimento, aumentando o tamanho do projeto

Portanto, conforme Keith [2010], as práticas ágeis não solucionam os problemas de desenvolvimento como num passe de mágica. Há muitos outros fatores envolvidos, sendo que o talento das pessoas é um fator crítico de sucesso, independentemente da metodologia ou processo utilizado. Adicionalmente, algumas dificuldades peculiares ao mercado de jogos, como a multidisciplinaridade, podem ser minimizadas, mas não eliminadas. É importante destacar que o objetivo do movimento ágil nunca foi produzir uma metodologia "perfeita", mas sim mostrar o melhor caminho.

No entanto, existem fatos que indicam que as práticas ágeis são mais adequadas para o desenvolvimento de jogos do que o processo tradicional utilizado (modelo em cascata). Apresentamos abaixo algumas práticas e atividades importantes no desenvolvimento de jogos que são apoiadas pelas práticas ágeis:

- **Encontrar a diversão o mais cedo possível**: jogos mais divertidos resultam em mais vendas e lucros. No entanto, encontrar a diversão do jogo é uma atividade difícil e que não pode ser realizada apenas com boa documentação [FULLERTON, 2008; GIBSON, 2007; KEITH, 2010]. Quanto mais cedo as ideias do jogo puderem ser testadas, mais cedo se encontrará a diversão do jogo. No modelo em cascata, a diversão é encontrada principalmente apenas no último terço do projeto, quando as partes são integradas e o jogo inteiro é testado. Ou seja, isso ocorre no pior momento possível, quando o projeto está perto de seu *deadline* [KEITH, 2010]. Em contraste com isso, as práticas ágeis incentivam a entrega constante de funcionalidades [AGILE ALLIANCE, 2006]. Adicionalmente, há a preocupação em desenvolver primeiro o que é mais importante. Versões jogáveis são lançadas periodicamente, e a cada iteração a diversão do jogo é questinada e, se necessário, o projeto é adaptado. A figura 2.2 constrasta a procura pela diversão entre projetos de jogos que utilizam o modelo em cascata e projetos que utilizam práticas ágeis.
- **Testes, reflexão e adaptação**: encontrar a diversão do jogo exige testes, reflexão e adaptação constantes [KEITH, 2010]. Segundo Schofield [2007], não é possível criar jogos divertidos sem adaptar a jogabilidade com base nos resultados obtidos até o momento. No entanto, testes, reflexão e adaptação são difíceis de serem aplicadas ao longo de todo o projeto no modelo em cascata. Em muitos casos, aplicar estas práticas num projeto guiado pelo modelo em cascata implicaria numa quebra do ciclo. Nas metodologias ágeis, por outro lado, estas práticas são extremamente incentivadas [AGILE ALLIANCE, 2006]. De fato, são práticas que fazem parte do núcleo dos métodos ágeis. Assim, aparentemente o desenvolvimento de jogos é inerentemente ágil [PETRILLO, PIMENTA, 2010].
- **Construir e utilizar o conhecimento**: algo fundamental para criar um jogo divertido é aproveitar ao máximo possível o conhecimento e experiência agregados durante o projeto [KEITH, 2010]. Ao desenvolver um jogo, estamos constantemente aprendendo onde está a diversão do jogo, quais são as *features* que mais agradam aos jogadores, quais erros não podemos repetir e assim por diante. No modelo em cascata, a utilização do conhecimento criado ao longo do projeto é difícil, pois as principais decisões são tomadas nas etapas iniciais e depois dificilmente são alteradas. Em contrapartida, o planejamento

ágil tem como foco construir conhecimento valioso e utilizá-lo com o objetivo de ajustar o plano para condizer com a realidade [KEITH, 2010].

- **Eliminar o disperdício**: com a preocupação constante em achar a diversão o mais cedo possível, muito trabalho é poupado. As *features* são desenvolvidas em iterações curtas e então testadas. As *features* que, após testadas, não agregarem valor ao jogo são imediatamente adaptadas ou canceladas. Em contraste, no modelo em cascata as necessidades de mudanças são detectadas tardiamente, quando muitos *assets* já foram criados, sendo que possivelmente vários deles terão de ser modificados ou nem mesmo serão utilizados [KEITH, 2010], aumentando o custo do projeto.

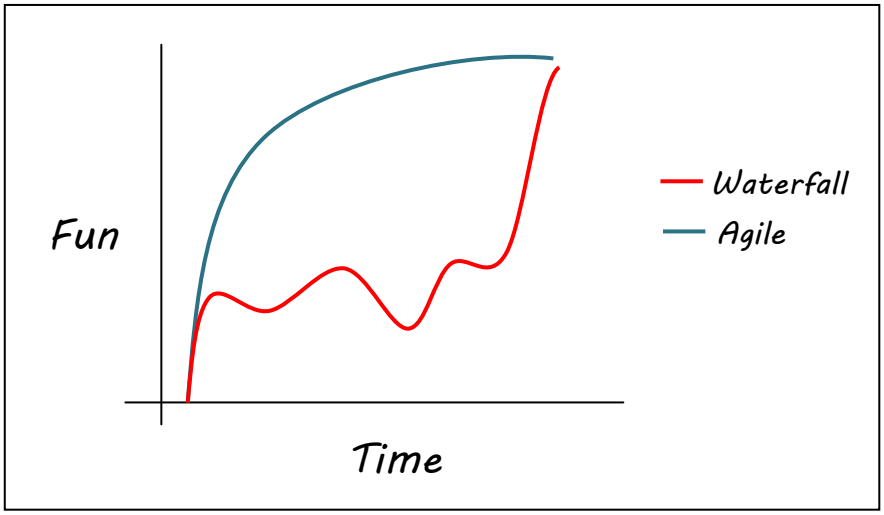


Figura 2.2. Encontrando a diversão do jogo ao longo do tempo [KEITH, 2010]

Existem alguns resultados concretos que apoiam a utlização de métodos ágeis no desenvolvimento de jogos. Em 2008, Petrillo [2008], analisou um conjunto de postmortens e identificou os problemas e as práticas ágeis mencionados nestes documentos. Como resultado da análise, conseguiu-se concluir que, na média, as equipes que adotaram mais práticas ágeis tiveram menos problemas do que as que não adotaram tais práticas. Resultados promissores em relação a métodos ágeis aplicados no desenvolvimento de jogos foram também obtidos por Keith [2010] e Gibson [2007].

2.2 Metodolologias ágeis aplicadas no desenvolvimento de jogos

Os métodos ágeis estão cada vez mais presentes nas empresas de desenvolvimento de *software* – e em muitas outras indústrias e disciplinas. Existem inúmeras referências bibiliográficas que abordam detalhadamente métodos ágeis como o *Extreme Programming*, *Scrum* e *Kaban*. Muitos autores abordam casos de sucesso, bem como relatos em que houveram dificuldades na aplicação da agilidade em projetos de desenvolvimento de *softwares*.

Infelizmente, com respeito a aplicação de agilidade no mercado específico de desenvolvimento de jogos eletrônicos, a literatura disponível é menos abundante.

Existem poucas obras de referência, e os casos de sucesso documentados são bastante raros. Essa lacuna é uma forte motivação para explorarmos de forma mais abrangente a agilidade em projetos de jogos, a fim de descobrirmos o pleno potencial dessa área de estudo.

Nesta seção, iremos abordar quatro metodologias ágeis ou parcialmente ágeis que foram aplicadas no desenvolvimento de jogos eletrônicos. As três primeiras são: *Game Unified Process*, *Extreme Game Development* e o *Scrum*. Por fim, vamos considerar o *Game Agile Methods Applied* (GAMA), foco principal deste trabalho.

2.2.1 *Game Unified Process*

O *Game Unified Process* (GUP) foi criado em 2003 por um gerente de projetos de jogos eletrônicos chamado Kevin Flood. O GUP foi apresentado através de um artigo submetido ao portal de desenvolvedores de jogos GameDev [FLOOD, 2003].

Flood percebeu muitos problemas no desenvolvimento de um projeto de jogo que ele gerenciou. Entre os problemas encontrados estão cronograma atrasado e constante pressão sobre o time de desenvolvimento. Segundo Flood, a razão por detrás destes problemas está no processo de desenvolvimento e gerenciamento utilizado, o *Waterfall* [FLOOD, 2003].

Um dos problemas do *Waterfall* indentificados por Flood é que o desenvolvimento segue um fluxo linear, sendo que a cada fase o jogo é observado por diferentes pessoas, e os diferentes tipos de testes são realizados somente nas fases finais do projeto. Neste ponto, frequentemente são detectadas necessidades de mudanças, que fazem com que o projeto precise retroceder para uma etapa que, por definição, não deveria ser revisitada. O fluxo proposto pela metodologia *Waterfall* é quebrado, resultando numa série de miniciclos não previstos originalmente [FLOOD, 2003].

Segundo Flood, outra característica do *Waterfall* que o torna inadequado para o desenvolvimento de jogos é a premissa de que as mudanças detectadas não serão drásticas por natureza. Em geral, esta premissa não condiz com a realidade do universo de desenvolvimento de jogos. Flood defende que o desenvolvimento de jogos é, por natureza, incremental [FLOOD, 2003].

Como alternativa ao *Waterfall*, Flood sugere a utilização híbrida de características do *Rational Unified Process* (RUP) e do *Extreme Programming* (XP). Os ciclos longos do RUP são combinados com os ciclos curtos do XP. A característica da documentação extensiva do RUP é utilizada junto com práticas do XP, como, por exemplo, os testes contínuos executados desde as fases iniciais do projeto.

Infelizmente, o autor não explica em detalhes como o *Game Unified Process* funciona. Aparentemente, o objetivo principal do GUP é chamar a atenção dos desenvolvedores para o fato de que o processo em cascata não é o melhor caminho, sendo que processos incrementais e ágeis podem ser alternativas interessantes e apropriadas.

2.2.2 *Extreme Game Development*

O *Extreme Game Development* (XGD) foi descrito em 2003 por Thomas Demachy através de um artigo submetido ao portal Gamasutra [DEMACHY, 2003]. Neste artigo,

o XGD é definido como um "método ágil para producão de jogos". Como o nome sugere, o *Extreme Game Development* está fortemente ligado ao amplamente conhecido método ágil *Extreme Programming* (XP). O XGD foi concebido no ambiente de desenvolvimento da empresa em que Demachy trabalhava na época, a Titus Interactive Studio, que então desenvolvia jogos eletrônicos.

Segundo Demachy, muitos times de desenvolvimento de jogos tem usado o *Extreme Programming* como sua metodologia padrão. No entanto, o XP possui uma limitação importante quando aplicado no universo dos jogos eletrônicos: ele foi criado principalmente para suprir as necessidades de programadores. Visto que um projeto de jogo eletrônico envolve pessoas de diversas disciplinas, viu-se a necessidade de adaptar o XP para que ele pudesse absorver esta pecularíedade dos projetos de jogos eletrônicos. Como resultado desta adaptação, surgiu o *Extreme Game Development* [DEMACHY, 2003].

No *Extreme Game Development*, existe a preocupação em diminuir problemas comumente encontrados em projetos de jogos eletrônicos, tais como desenvolvimento constantemente atrasado e mudanças de requisitos por parte dos *publishers*. Para alcançar este objetivo, o XGD baseia-se fortemente nos princípios ágeis publicados no Manifesto Ágil [AGILE ALLIANCE, 2006] e nos quatro valores defendidos no *Extreme Programming*: simplicidade, comunicação, *feedback* e coragem [BECK, 1999].

Seguem abaixo algumas características e práticas do XGD:

- O *publisher* deve estar presente no local de trabalho do time de desenvolvimento
- *Milestones* curtos, de seis semanas, onde é feito um planejamento e escolha das *features* mais importantes a serem desenvolvidas primeiro
- Ciclos de desenvolvimento de três semanas, onde as tarefas são definidas, estimadas e desenvolvidas
- Integração contínua suportada por ferramentas apropriadas
- Todos são parte do time, logo todos são responsáveis pelo jogo
- Testes automatizados
- Versionamento
- Utilização da *Unified Modeling Language* (UML)

Um artefato previsto no *Extreme Game Development* é o "XGD *dashboard*". Este é um quadro atualizado a cada fim de *milestone*, quando o gerente do projeto faz uma reflexão com base no *feedback* recebido até o momento. Nele, o gerente coloca cinco práticas e ferramentas que estão sendo usadas, bem como cinco que gostaria de usar a partir do próximo *milestone*. Para cada uma das práticas ou ferramentas que estão sendo utilizadas, dá-se uma pontuação de zero a cinco. O histórico da pontuação indica se o XGD está sendo utilizado plenamente, bem como se a equipe está motivada em aplicá-lo, ou se o XGD não é a metodologia certa para o time de desenvolvimento. Desta forma, o XGD prevê a constante reflexão e aprimoramento do processo, embora centralizados em uma única pessoa [DEMACHY, 2003].

Há pelo menos um caso documentado a respeito da utilização do XGD. Andy Krouwel [2010], desenvolvedor de jogos, relata a experiência de aplicar o XGD

conforme vivenciado na empresa onde trabalha. Segundo ele, o principal benefício de usar o XGD é o fato de sempre haver uma versão do jogo funcionando. Isso permite visualizar o progresso do projeto e se manter focado em torná-lo mais completo e divertido, sem desconsiderar os prazos.

2.2.3 *Scrum*

O *Scrum* é um processo ágil que foi formalizado a cerca de dez anos. Desde então, a aceitação e utilização do *Scrum* cresceu de forma extremamente rápida, sendo que atualmente muitas companhias de diversos tamanhos adotam o *Scrum* como sua metodologia padrão [KNIBERG, 2007].

Dentre os processos ágeis existentes, possivelmente o *Scrum* seja o mais utilizado no escopo de desenvolvimento de jogos. Um dos motivos pelos quais isso acontece é o fato de que o *Scrum* não foi concebido para atender apenas as necessidades de projetos de desenvolvimento de *software*, mas sim as de qualquer projeto complexo, de qualquer área, especialmente os que envolvem coisas inovadoras [KEITH, 2010; SCRUM ALLIANCE, 2011]. Desta forma, pode-se dizer que o *Scrum* se encaixa de modo natural com o processo de desenvolver jogos, que são, em sua maioria, projetos complexos e multidisciplinares.

Adicionalmente, os princípios e práticas incentivadas pelo *Scrum*, como adaptação, reflexão e constante aprimoramento permitem que o processo seja ajustado de forma a atender mais plenamente as necessidades específicas de projetos de desenvolvimento de jogos.

Uma das referências em aplicação do *Scrum* no desenvolvimento de jogos é Keith [2010], que desde 2005 tem ajudado empresas de jogos eletrônicos a adotarem o *Scrum*, com resultados excelentes. Em 2010, Keith publicou o livro *Agile Game Development With Scrum* [KEITH, 2010], onde é abordado como aplicar o *Scrum* no desenvolvimento de jogos eletrônicos.

Segundo Keith [2010], algumas atividades, práticas e princípios incentivados pelo *Scrum*, valiosos quando aplicados no desenvolvimento de jogos são:

- **Planejamento ágil**: o planejamento deve ser distribuído ao longo de todo o projeto e não concentrado em uma etapa específica [KEITH, 2010]. As *features* mais importantes são planejadas em mais detalhes; o que não é importante no momento pode ser planejado de forma completa mais tarde. Versões jogáveis são mais importantes do que documentação extensa [AGILE ALLIANCE, 2006]. Deve-se evitar planejar e estimar coisas que talvez nem façam parte do jogo [AMBLER, 2004].
- **Priorização:** ao longo do projeto, são realizadas várias priorizações que permitem ao time de desenvolvimento trabalhar sempre no que é mais importante primeiro. Ou seja, o foco sempre está em desenvolver o que é mais divertido [KEITH, 2010].
- **Iterações rápidas**: os *sprints*, cujas durações normalmente são de poucas semanas, incentivam a entrega constante de funcionalidades [HIGHSMITH, 2002]. Isso permite visualizar de maneira mais clara o progresso do jogo. A cada

fim de *sprint*, incentiva-se que as funcionalidades sejam integradas de forma a produzir uma nova versão do jogo [KEITH, 2010].

- **Testes**: o jogo é testado em vários momentos por diferentes pessoas, resultando em um *feedback* valioso que pode ser tomado como base para futuras decisões [GIBSON, 2007].
- **Conhecimento, experiência e adptabilidade**: o conhecimento e experiência agregados durante o desenvolvimento do jogo são usados para adaptá-lo ao que é mais importante, isto é, a diversão [KEITH, 2010].
- **Contato direto**: o *Scrum* incentiva a comunicação face a face sempre que possível e necessário [HIGHSMITH, 2002; SCRUM ALLIANCE, 2011]. Em projetos de jogos, discussões periódicas a respeito do conceito do jogo, da diversão e dos resultados dos testes são fundamentais. Além disso, a reunião em pé, realizada diariamente, permite que cada membro da equipe relate o andamento de suas tarefas.

Resultados promissores sobre a aplicação do *Scrum* no desenvolvimento de jogos foram obtidos por Gibson [2007] num estudo de caso. Neste experimento, que durou alguns meses, *designers*, programadores e jogadores foram recrutados para ajudarem no desenvolvimento de um jogo utilizando o *Scrum* como metodologia. A cada duas semanas, uma nova versão do jogo foi entregue e submetida a testes dos jogadores. O *feedback* gerado pelos testes era então discutido e utilizado para adaptar o jogo.

Segundo Gibson [2007], o *Scrum* se mostrou apropriado para o desenvolvimento de jogos. A principal contribuição do *Scrum* para o sucesso do experimento foi a entrega periódica de versões submetidas a testes dos jogadores. Isso permitiu que os *designers* adaptassem o jogo conforme o que era considerado mais divertido pelos jogadores. Como resultado, a cada versão entregue a diversão do jogo era maior, conforme opinião dos próprios jogadores.

2.2.4 GAMA

Em 2008, Petrillo [2008], então aluno de mestrado do curso de Ciência da Computação do Instituto de Informática da Universidade Federal do Rio Grande do Sul, sob orientação do professor Dr. Marcelo Soares Pimenta, fez um estudo sobre Práticas Ágeis no Processo de Desenvolvimento de Jogos Eletrônicos [PETRILLO,2008]. Este estudo culminou num conjunto de práticas ágeis agrupados em forma de metodologia específica para o desenvolvimento de jogos, chamada *Game Agile Methods Applied* (GAMA) [PETRILLO,2008].

Em seu trabalho, Petrillo [2008] analisou detalhadamente um conjunto de postmortens. Durante esta análise, foram identificados diversos problemas recorrentemente citados pelos autores dos postmortens [PETRILLO, PIMENTA, TRINDADE, DIETRICH, 2009]:

- Escopo irreal ou ambicioso
- *Features* acrescidas tardiamente
- *Features* retiradas durante o desenvolvimento
- Problemas na fase de projeto

- Atraso ou otimismo no cronograma
- Problemas tecnológicos
- *Crunch time*[1]
- Falta de documentação
- Problemas de comunicação
- Problemas com ferramentas
- Problemas na fase de teste
- Montagem da equipe
- Grande número de defeitos
- Perda de profissionais
- Orçamento extrapolado

De acordo com Petrillo [2008] e outros autores, como Flood [2003] e Keith [2010], muitos destes problemas estão relacionados com o processo tradicional de desenvolvimento de jogos, ou seja, o desenvolvimento em cascata. Diante desta perspectiva, Petrillo [2008] analisou quais práticas das principais metodologias ágeis (Desenvolvimento Enxuto de *Software*, *Extreme Programming*, Modelagem Ágil e *Scrum*) poderiam eliminar ou minimizar estes problemas, conforme ilustrado na figura 2.3.

Figura 2.3. Métodos Ágeis que formam o GAMA [PETRILLO, 2008]

Com base nos resultados obtidos através da análise realizada, Petrillo [2008] selecionou e agrupou algumas práticas ágeis e elaborou uma metodologia específica para o desenvolvimento de jogos eletrônicos, o *Game Agile Methods Applied* (GAMA). Esta seleção de práticas ágeis aplicáveis ao desenvolvimento de jogos levou em consideração quatro critérios principais:

- Aderência das boas práticas já adotadas em projetos de jogos aos métodos ágeis
- Correlação entre os problemas encontrados e as boas práticas levantadas
- Práticas que tratam dos problemas da indústria de jogos

[1] *Crunch time* é um termo usado no basquete que se refere aos últimos minutos do jogo, quando tipicamente os times lutam intensamente pela vitória. Na indústria de jogos, esse termo é utilizado para se referir aos períodos de sobrecarga de trabalho, especialmente comuns nas semanas que antecedem o prazo final da entrega do projeto [PETRILLO, PIMENTA, TRINDADE, DIETRICH, 2009].

- Grau de maturidade necessária para a implantação das práticas ágeis

Segundo Petrillo [2008], o GAMA, como um conjunto de práticas, pode ser organizado para a execução básica de um projeto de desenvolvimento de jogos, servindo de referência para a definição das atividades e elaboração dos artefatos.

O GAMA é divido em duas fases principais:

- **fase exploratória**
- **fase de execução**

Transcrevemos abaixo a descrição das fases e das etapas do GAMA tais quais definidas originalmente por Petrillo [2008].

A **fase exploratória** compreende a etapa de projeto na qual os conceitos básicos do jogo são definidos, geralmente iniciada com uma ideia ou inspiração de como o jogo pode ser feito. Essa fase é formada pelas seguintes atividades e seus respectivos produtos:

- **Elaborar conceito do jogo**: é a fase durante a qual os principais elementos que irão compor o jogo são definidos, como os roteiros, os personagens e os ambientes. Essa fase deve seguir as ideias de McShaffry [2003], criando um documento do jogo que descreva apenas as linhas gerais da jogabilidade, concentrando os esforços na definição da jogabilidade em alto nível. Também são tomadas as decisões de arquitetura e de tecnologia a serem adotadas. Produto: documento do jogo.
- **Determinar atores e estórias de usuário:** a partir do documento do jogo, são definidos os atores e estórias de usuário que formam o jogo, indicando seus relacionamentos. Produto: diagrama de casos de uso.
- **Elaborar estórias de usuário:** mapeados os casos de usos, eles são especificados em estórias de usuário, em cartões do tipo fichas pautadas, nos quais são definidas as ações, os fluxos, as interações e os comportamentos dos atores no jogo. Nesse processo, membros de toda a equipe devem participar, especialmente, os roteiristas e a equipe de arte. Dessa atividade, novas estórias de usuário podem surgir, complementando o diagrama de casos de uso. No verso desses cartões são elaborados os esboços de interface. Também são elaborados os casos de teste para cada cartão. Produtos: estórias de usuário e casos de teste.
- **Planejar iterações:** com as estórias detalhadas, é feita uma reunião de planejamento das iterações, na qual as estórias são priorizadas, além de classificadas segundo sua complexidade e risco. Produto: backlog do produto.

A **fase de execução** compreende o conjunto de atividades de planejamento e execução das tarefas para a construção do jogo propriamente dito. Essa fase é formada pelas seguintes atividades e seus respectivos produtos:

- **Definir e estimar as tarefas**: as estórias priorizadas são selecionadas para a formação do *sprint*. Para cada estória, são determinadas as tarefas a serem realizadas para a implementação, tanto artística quanto de programação. As tarefas são escritas em *postits* e estimadas em pontos, utilizando-se a técnica

de *planning poker* [KNIBERG, 2007]. Produtos: Backlog do *sprint* e cartões (*postits*) de tarefas.

- **Reunião de pé**: cada dia é iniciado com uma reunião rápida e de pé, na qual todo o integrante da equipe relata o que fez no dia anterior e o que pretende realizar. Ao final, os participantes assumem as tarefas, realocando-as no quadro e atualizando o gráfico de *burndown*. Produtos: quadro de tarefas e gráfico *burndown*.
- **Sessão de modelagem ágil**: caso necessário, na sessão de modelagem ágil são modelados os diagramas e esboços para a execução das tarefas. Produtos: diagramas ou esboços de solução.
- **Sessão de implementação**: durante a implementação das tarefas são aplicadas as práticas de desenvolvimento guiado por teste[1], refatoração e integração contínua, tanto de código fonte quanto de arte, som ou outros artefatos do jogo. Produtos: testes unitários, código fonte, arte, som, etc.
- **Executar atividades de qualidade**: após a integração dos módulos, o resultado do *software* deve ser testado por uma equipe especializada, realizando-se testes caixa preta que atendam aos casos de teste definidos para as estórias; testes de cobertura; testes de integração; e, caso necessário, testes de carga [PRESSMAN, 2006]. Os defeitos e aperfeiçoamentos podem ser relatados em uma ferramenta de *bug tracking* e lançados como novas tarefas no próprio *sprint* ou no seguinte. Produto: Relatório de *bugs* e *postits*.
- **Integrar versão**: se todas as estórias implementadas atendem aos casos de teste ou atingiu-se o prazo final do *sprint*, o jogo é integrado em uma versão para demonstração. A versão resultante do primeiro *sprint* pode ser utilizada no Greenlight Process[2]. Versõesintegradas em novos *sprint* podem ser rotulados, tradicionalmente, como Alfa[3] ou Beta[4]. Produto: versão do jogo.
- **Reunião de retrospectiva**: ao final do *sprint* é realizada uma reunião, com toda a equipe, objetivando refletir sobre o processo de trabalho realizado, propondo aperfeiçoamentos para a próxima iteração. Produto: propostas de aperfeiçoamento do processo.
- **Entregar a versão final**: a última atividade consiste em empacotar a versão final para distribuição, seja através de mídias ou pela Internet.

Uma característica básica do GAMA é o gerenciamento de projetos baseado no *Scrum* [HIGHSMITH, 2002]. Um dos motivos que levou Petrillo [2008] a adotar essa abordagem foram os bons resultados obtidos por Keith [2010].

---

[1] O desenvolvimento guiado por teste é conhecido em inglês como Test Driven Development (TDD).

[2] O Greenlight Process é uma reunião de apresentação de um protótipo ou de uma versão inicial do jogo para que os investidores possam decidir se o projeto será financiado ou abandonado [GERSHENFELD; LOPARCO; BARAJAS, 2003].

[3] Alfa é a primeira versão completamente jogável de um jogo [GERSHENFELD; LOPARCO; BARAJAS, 2003].

[4] Beta é a versão “completa”

Na figura 2.4, apresentamos as etapas do GAMA representadas através de um diagrama de atividades.

Após a elaboração do GAMA, Petrillo [2008] aplicou-o num estudo de caso. O estudo de caso foi restrito ao meio acadêmico e teve várias limitações. Contudo, os resultados obtidos foram animadores, sendo que o projeto onde o GAMA foi utilizado produziu resultados significantemente melhores do que o projeto onde o processo em cascata foi utilizada.

No próximo capítulo, iremos analisar o GAMA e propor estensões que o tornem ainda mais completo e adequado as necessidades de projetos de desenvolvimento de jogos eletrônicos.

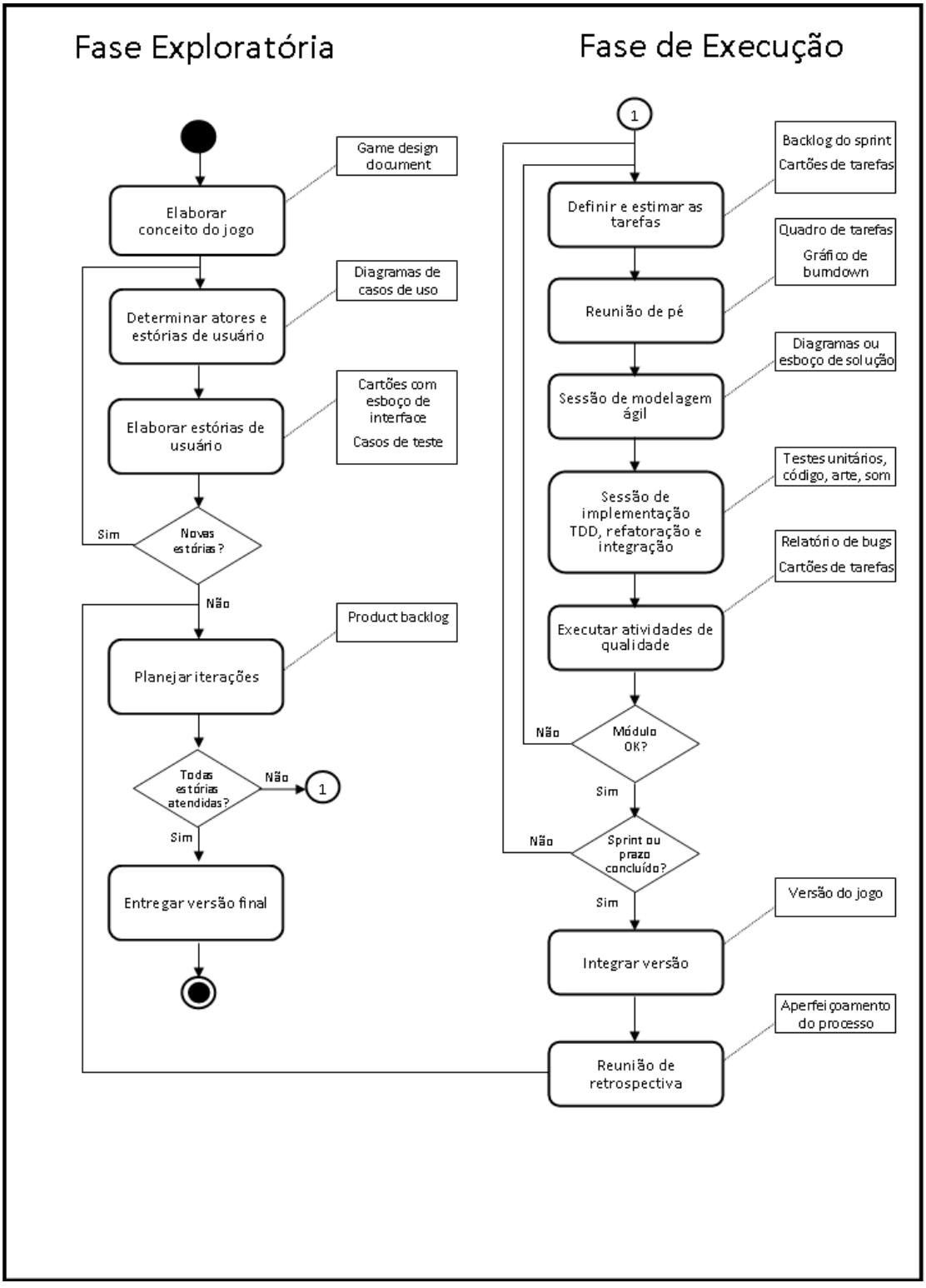


Figura 2.4. Diagrama de atividades do GAMA

3. ANALISANDO E ESTENDENDO O GAMA

Neste capítulo, será apresentada uma análise sobre o GAMA. O objetivo desta análise é identificar possíveis aprimoramentos que tornem a metodologia mais completa em relação as necessidades dos projetos de jogos eletrônicos. Na sequência, serão apresentadas algumas propostas de mudanças ao GAMA a fim de incorporar os aprimoramentos identificados na análise, resultando numa versão estendida.

3.1 Analisando o GAMA

Em 2008, ano em que o GAMA foi elaborado, havia pouca literatura a respeito de métodos ágeis aplicados a jogos. De fato, mesmo atualmente ainda estamos descobrindo os benefícios, dificuldades e desafios de aplicar métodos ágeis no desenvolvimento de jogos. Por esse motivo, pode-se dizer que o GAMA é uma metodologia inovadora.

Tendo em vista a abordagem inovadora do GAMA em relação ao desenvolvimento de jogos quando comparada as metodologias tradicionais, é natural que hajam aspectos que possam ser aprimorados ou tornados mais práticos. Além do mais, processos e metodologias levam tempo para atingirem a maturidade, sendo que a filosofia ágil incentiva a constante reflexão e aprimoramento dos mesmos [AGILE ALLIANCE, 2006].

Analisando o GAMA, a luz das boas práticas de desenvolvimento e das necessidades específicas da indústria de jogos, identificamos os seguintes pontos que podem ser incorporados no GAMA: prototipação, teste com jogadores finais[1] e maior suporte a adaptabilidade. Tais aprimoramentos tem como objetivos principais auxiliar a encontrar e potencializar a diversão do jogo. Vamos considerar detalhadamente cada um destes pontos.

3.1.1 Prototipação

Um dos principais fatores de sucesso de um jogo é até que ponto ele diverte as pessoas. De fato, esse é o principal objetivo dos jogos: criar emoções que levem, de uma maneira ou outra, a diversão.

Contudo, o conceito diversão não é algo simples de ser definido. Envolve a experiência que emerge da interação do jogador com o jogo. Consequentemente, a diversão é algo abstrato que varia de pessoa para pessoa [GIBSON,2007]. Além do

[1] Neste trabalho, jogadores finais se referem a potenciais clientes do jogo

mais, as pessoas se divertem com coisas diferentes; o que é divertido para um pode não ser para outro. A diversão é um conceito subjetivo, contextual, que depende de muitos fatores e que é pessoal [FULLERTON, 2008].

Dessa forma, criar algo divertido não é um processo linear e envolve elementos que ainda não entendemos plenamente. A diversão em jogos pode ser considerada como um requisito não-funcional que é difícil de ser contemplado [CALLELE, NEUFELD, SCHNEIDER, 2005]. Consequentemente, o processo de tornar um jogo divertido não é direto nem procedural. Não existe um método que garanta que uma ideia, quando colocada num jogo, produzirá diversão aos jogadores. Ou seja, a única forma de saber com alguma certeza se uma ideia agrega valor ao jogo é testando-a, possivelmente através de protótipos.

De acordo com Schofield [2007], para encontrar uma jogabilidade divertida que combine com o conceito do jogo, os *designers* precisam mudar pontualmente a direção do mesmo a medida que aprendem com o processo. Tal aprendizado pode ser potencializado com o uso de protótipos e o consequente *feedback*.

A utilização de protótipos, conforme Fullerton [2008], permitem que as principais *features* do jogo, como ideias conceituais, jogabilidade, arte, controles e interface passem por um processo de teste e validação antes de serem tomadas decisões definitivas. Os protótipos podem ser usados, por exemplo, para ajudar a esclarecer aspectos do jogo em que há grande incerteza ou a comparar soluções. Assim, os protótipos podem ser de muita utilidade para responder a questões importantes, tais como: será que essa ideia deixará o jogo mais divertido? Qual destas duas mecânicas é mais apropriada? Este efeito visual pode ser implementado na tecnologia que temos a disposição? Resumindo, a prototipação permite esclarecer o que realmente funciona antes de fazer o *design* final do jogo.

Considere, por exemplo, que o *Game Design Document* de um *first-person shooter*[1] mecione o uso de objetos físicos que quebram ao serem atingidos por tiros. Uma ideia legal, mas será que o custo de desenvolver essa *feature* compensa? Será que o engine físico suporta geometria destrutível? Quais são os riscos envolvidos? Seria prudente prototipar essa *feature* para responder estas questões antes de utilizá-la de forma mais plena no *design* do jogo. Portanto, a prototipação é especialmente importante quando o *designer* desconhece o valor, custo e risco de uma *feature* e precisa de mais informações para tomar uma decisão [KEITH, 2010].

Quando um protótipo está sendo implementado, em geral deve-se focar em esclarecer uma questão importante do jogo. Nesse estágio, não há preocupação em deixar o jogo bem acabado, bonito e livre de *bugs*. Isso permite que as ideias sejam testadas e validadas mais rapidamente. De fato, um dos pontos importantes a respeito de protótipos é que eles não devem levar muito tempo para serem desenvolvidos [FULLERTON, 2008; KEITH, 2010].

Quando uma *feature* importante sobre a qual há grande incerteza é atacada num protótipo, o *feedback* gerado é a base para a tomada de decisões sobre tal *feature*.

[1] *First person shooter* (FPS) é um gênero de jogos que centra a jogabilidade em torno de armas e combates através da perspectiva da primeira pessoa perspectiva, ou seja, o jogador experimenta a ação através dos olhos de um protagonista [WIKIPEDIA, 2011b].

Caso o *feedback* seja positivo, a solução implementada no protótipo é refinada e passa a ser definitiva. Caso contrário, deve-se prototipar uma solução alternativa ou cancelar a *feature* em questão. Quando as ideias principais do jogo são prototipadas e mostram ser interessantes para os jogadores, então o *design* do jogo é sólido [FULLERTON, 2008].

A capacidade de detectar mudanças necessárias através da prototipação é extremamente importante. Conforme mostrado na figura 3.1, quanto mais cedo for detectada uma mudança de requisitos, menos dinheiro, tempo e esforço serão necessários para fazer com que o projeto seja ajustado as novas necessidades [BOEHM, 1981; KEITH, 2010]. Deste modo, a prototipação, quando bem utilizada, permite potencializar os lucros e poupar recursos.

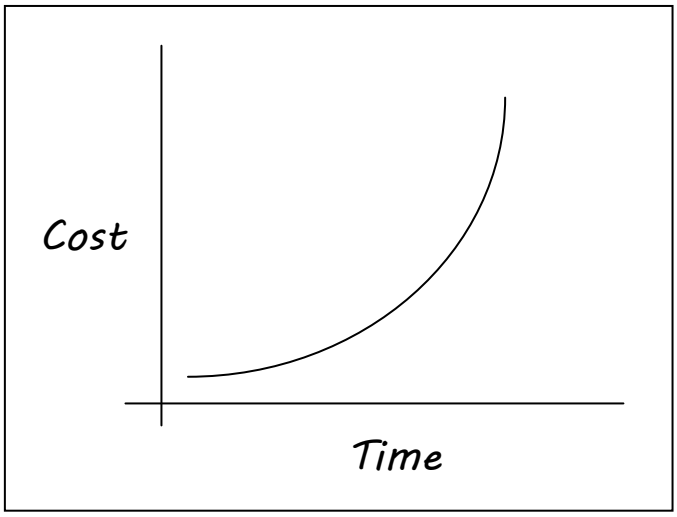


Figura 3.1. Custo de mudanças ao longo do tempo

Portanto, o bom uso de protótipos pode auxiliar os *designers* a encontrar a diversão, além de esclarecer questões importantes de forma rápida, permitindo antecipar problemas e rever o *design* do jogo antes que seja tarde demais. Acreditamos que estes benefícios justifiquem a explicitação da prototipação na metodologia de desenvolvimento.

3.1.2 Teste com jogadores finais

Como já mencionado, tornar um jogo divertido é algo complexo, que depende, entre outros fatores, de boas ideias conceituais e de uma mecânica que permita ao jogador explorar tais ideias de forma eficiente. Tal complexidade justifica o uso de prototipação, conforme analisado na seção anterior.

Contudo, apenas a prototipação junto com testes da equipe de desenvolvimento e *designers* pode não ser suficiente para validar ou não algumas ideias e mecânicas. Na realidade, testes executados pelas pessoas envolvidas no projeto do jogo apresentam algumas armadilhas:

- **Os desenvolvedores conhecem o jogo**: os desenvolvedores, por conhecerem o jogo, ao testá-lo, podem não perceber problemas de usabilidade, jogabilidade e outros. Por exemplo, pode ser que haja uma parte do jogo na qual a solução

para um problema seja demasiadamente oculta ou difícil, fazendo com que jogadores finais não consigam progredir no jogo. Nesse caso, um desenvolvedor, por saber a solução do problema a *priori*, não teria tais dificuldades e por esse motivo não conseguiria detectar que essa parte do jogo precisa de melhorias. O mesmo pode acontecer com mecânicas muito complexas para jogadores finais, mas que podem parecer apropriadas para um desenvolvedor que está, naturalmente, bastante treinado.

- **Os desenvolvedores trabalham no jogo**: o grande conhecimento que os desenvolvedores possuem sobre o jogo pode impedi-los de determinar se o jogo está divertido ou não. Após trabalhar meses num projeto de jogo, 8 horas (ou mais) por dia, não é de admirar que um desenvolvedor enjoe do jogo que está desenvolvendo[1]. O contato constante com o jogo em desenvolvimento pode levar o desenvolvedor a achar pouco divertidas e enjoativas ideias que, quando testadas por jogadores finais, podem ter um resultado positivo. Detalhes técnicos como estilo gráfico e trilha sonora também podem se tornar estafantes para um desenvolvedor, embora sejam agradáveis para um jogador final, cujo contato com o jogo é menos intenso.
- **Os *designers* são apaixonados por suas ideias**: o teste e validação por parte dos *designers* é considerado muito importante, visto que eles, mais do que ninguém, possuem uma visão completa do jogo. No entanto, pode acontecer que um *designer* fique tão apaixonado e emocionado com suas próprias ideias contidas no jogo que sua habilidade de analisá-las criticamente fique comprometida. Nesse caso, ele corre o risco de desperceber que uma ideia é fabulosa para ele, mas talvez não o seja para quem mais importa, os jogadores [GIBSON, 2007].

Os motivos acima nos levam a crer que os testes da equipe de desenvolvimento não são suficientes para testar todos os aspectos do jogo, em especial a diversão e jogabilidade. Consequentemente, prever testes com jogadores finais na metodologia de desenvolvimento pode ser determinante para o sucesso de um jogo. Além disso, testes com usuários são muito incentivados pelas práticas ágeis, a ponto de ser uma das práticas principais do *Extreme Programming* [BECK, 1999].

Nos projetos de jogos eletrônicos, os testes com jogadores tem como objetivo principal capturar *feedback* útil a fim de identificar se o jogo está divertido. Além disso, através dos testes com jogadores pode-se avaliar se o jogo está ficando como o *designer* pretende [FULLERTON, 2008].

Adicionalmente, testes com jogadores podem auxiliar a minimizar ou eliminar um problema que ocorre frequentemente em projetos de jogos eletrônicos, a saber, a tendência dos desenvolvedores, de forma inconsciente, introduzirem no jogo suas próprias visões e ideias e esquecer de olhar o jogo sob o ponto de vista do jogador [FULLERTON, 2008]. É importante ressaltar que os testes com jogadores não substituem os testes da equipe de desenvolvimento e dos *designers*. Todos eles tem

[1] Certa vez assisti a uma palestra de um empregado de alto escalão da Crytec, onde ele disse que um desenvolvedor não deve trabalhar numa empresa que produza jogos que ele gosta, porque neste caso, inevitavelmente, este irá acabar odiando os jogos que antes lhe davam prazer.

seu valor, e juntos produzem um *feedback* extremamente importante para o sucesso do jogo.

3.1.3 Maior suporte a adaptabilidade

A adaptabilidade, as vezes chamada também de flexibilidade, é uma das principais características das metodologias ágeis, confome pode-se notar pelas referências abaixo citadas:

> *"We value responding to change over following a plan."* [AGILE ALLIANCE, 2006]
>
> *"We follow this principles: … welcome changing requirements, even late in development....."* [AGILE ALLIANCE, 2006]
>
> *"Scrum is a simple 'inspect and adapt' framework."* [SCRUM ALLIANCE, 2011]

Segundo [KEITH, 2010], as práticas ágeis permitem que se aprenda, ao longo do desenvolvimento do jogo, o que é divertido, o que é possível e como desenvolver. O conhecimento e experiência adquiridos a medida que o projeto é executado não devem ser desconsiderados. Ambos são importantes para adaptar o conceito do jogo ao que é mais divertido, factível e com melhor custo-benefício.

Vamos considerar novamente o exemplo do jogo do gênero *first-person shooter* cujo *design* prevê objetos físicos que quebram ao serem atingidos por tiros. A medida que essa *feature* é desenvolvida, pode ser que os programadores tenham mais trabalho do que o previsto e a *feature* seja muito custosa de implementar. Adicionalmente, os *designers* podem se dar conta de que essa *feature* não será impactante no que diz respeito a agregar diversão ao jogo. Após estas considerações, a equipe pode decidir cancelar a *feature*, ou implementar uma versão mais simples, como, por exemplo, propriedades físicas apenas nos principais objetos do cenário. Dessa forma, o jogo é adaptado considerando fatores como diversão e custo-benefício.

Conforme [SCHOFIELD, 2007], adaptar o jogo não é uma opção, é uma necessidade. É principalmente durante a execução do projeto, com entrega de versões jogáveis e testes que se descobre onde está a diversão do jogo. O conceito do jogo deve ser adaptado para explorar ao máximo as *features* que são mais divertidas. Isso não significa necessariamente que o conceito do jogo deva ser significamente alterado. Essa adaptação pode ser bem mais simples, como, por exemplo, priorizar *features* que antes eram consideradas menos divertidas e, consequentemente, menos prioritárias, mas que agora descobriu-se que são, na realidade, muito interessantes para os jogadores.

Um motivo adicional que justifica a importância da adaptabilidade é o fato de que a documentação inicial do jogo nunca é completa nem definitiva. Os *designers*, por mais capacitados e experientes que sejam, não conseguem prever todos os detalhes que farão do jogo um sucesso. Sempre haverá dúvidas, incertezas e aspectos não explorados. As respostas emergem junto com o progresso do jogo.

Portanto, não é de admirar que [KEITH, 2010] afirme que a diversão do jogo não pode ser encontrada somente com base no que diz o *Game Design Document*. De forma similar, os aspectos técnicos e de arquitetura colocados no GDD dificilmente são

aqueles que realmente precisamos e usamos no fim das contas. Keith [2010] exemplifica tais considerações com um *Game Design Document* de um jogo no qual ele participou do projeto. Neste GDD, estavam prescritos, não somentes as armas que o jogador teria a disposição, mas também a quantidade de munição máxima de cada arma, bem como o número de balas em cada cartucho. Este nível de detalhamento logo no início do projeto do jogo não traz benefícios e, potencialmente, pode até atrapalhar. Detalhes como o acima precisam ser mitigados durante a implementação do jogo. As decisões devem ser tomadas apenas quando há conhecimento suficiente para isso, implicando numa adaptabilidade natural a medida que as incertezas vão sendo diluídas [KEITH, 2010].

A importância da adaptabilidade foi constatada por [GIBSON, 2007], que fez um experimento em que um jogo foi desenvolvido utilizando o *Scrum*. Gibson afirma que um dos principais benefícios percebidos por adotar uma metodologia ágil foi a capacidade de ajustar, ou adaptar, as ideias do jogo tendo como base principalmente o *feedback* dos jogadores.

Com base nestas considerações, vê-se que não é apropriado comprometer o jogo com o *design* original até o final. Contudo, é preciso haver cuidado para que as mudanças durante o projeto não transformem o jogo originalmente planejado em outro completamente diferente. Um jogo de corrida não deve ser gradativamente alterado e se tornar um jogo de estratégia. A ideia da adaptabilidade aplicada em projetos de jogos é, principalmente, refinar o *Game Design Document* com base no conhecimento acumulado durante o progresso do desenvolvimento do jogo [KEITH, 2010].

As figuras que seguem ilustram o que frequentemente acontece com projetos tradicionais e em projetos em que a agilidade e a adaptabilidade estão presentes. Na figura 3.2, que exemplifica um projeto *waterfall*, o planejamento feito nas etapas iniciais é seguido até as etapas finais, quando, possivelmente, descobre-se (tardiamente) que há coisas que devem mudar para que o objetivo do projeto seja alcançado. Na figura 3.3, que exemplifica um projeto ágil com suporte a adaptabilidade, a medida que a necessidade de mudanças são detectadas, o projeto é adaptado, fazendo com que o desenvolvimento sempre seja feito na direção correta.

Tendo em mente as justificativas apresentadas, concluimos que a adaptabilidade, baseada no conhecimento que emerge durante o progresso do projeto do jogo e no *feedback* proveniente dos vários testes (*designers*, desenvolvedores e jogadores) e da prototipação é uma caracterísitca importante que deve fazer parte da metodologia de desenvolvimento.

O GAMA, apesar de ser uma metodologia que adota práticas ágeis, possui um suporte limitado no que diz respeito a adaptar o jogo ao longo do projeto. Nas primeiras etapas da fase exploratória são definidos o conceito do jogo e as estórias de usuário. Após isso, a metodologia prevê um ciclo entre planejamento das iterações, ou *sprints*, e implementação, sendo que não se prevê a alteração, ou adaptação, do conceito do jogo a medida que o projeto evolui. Pode-se dizer que a característica da adaptabilidade está implícita ou mesmo oculta. Visto que esta qualidade é fundamental no processo de desenvolvimento de jogos, consideramos que é

importante deixá-la mais visível, bem como tornar mais claro de que forma ela deve ser aplicada em projetos de jogos eletrônicos.

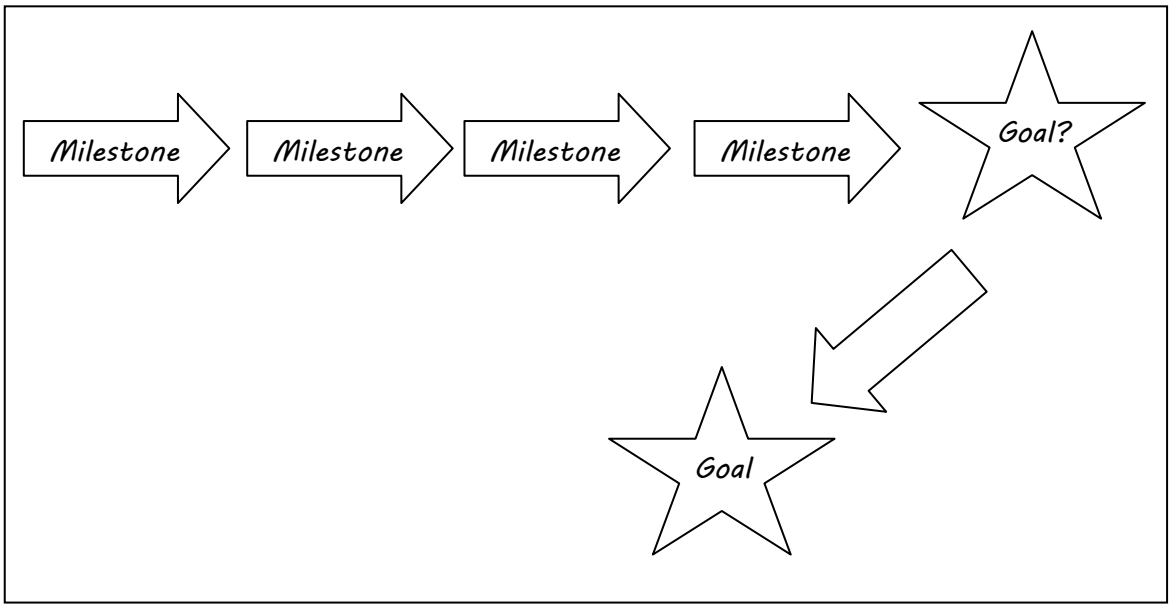


Figura 3.2. Projetos sem suporte a adaptabilidade [KEITH, 2010]

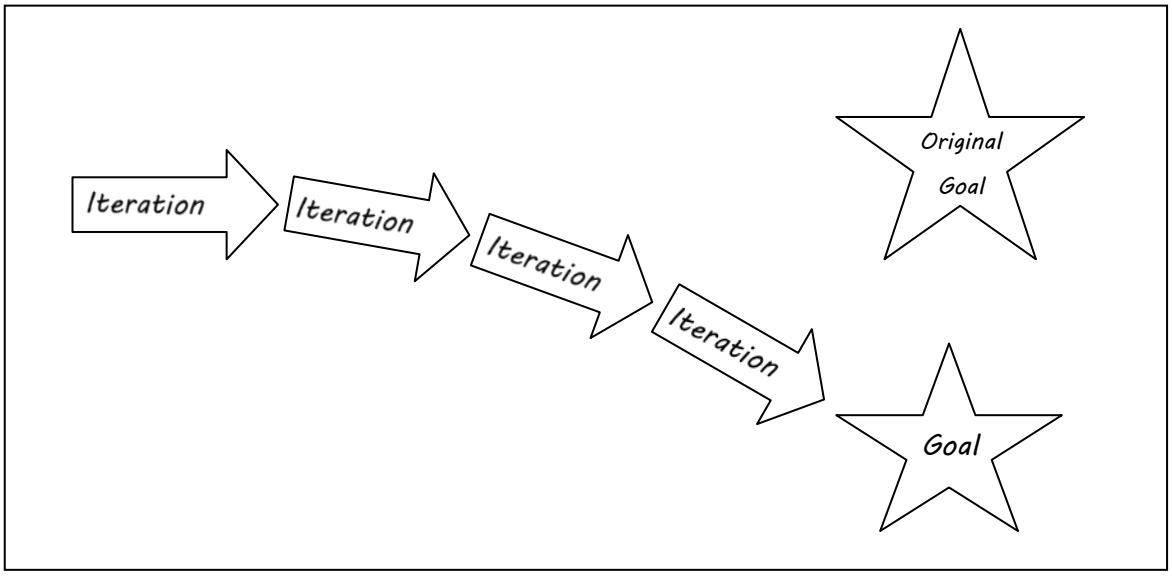


Figura 3.3. Projetos com suporte a adaptabilidade [KEITH, 2010]

3.2 Estendendo o GAMA

Como resultado da análise apresentada na seção anterior, identificamos e justificamos três aprimoramentos principais que podem ser incorporados ao GAMA: prototipação, testes com jogadores e maior suporte a adaptabilidade. Apresentamos abaixo as etapas e o diagrama de atividades do GAMA estendido para refletir tais aprimoramentos.

A **fase exploratória**, cujos objetivos principais são definir o conceito do jogo e planejar os *releases*, é composta pelas seguintes etapas:

- **Elaborar conceito do jogo**: o *product owner*, os *designers* e os *stakeholders* se reúnem para definir os principais requisitos do jogo. São definidos, em alto nível, os roteiros, personagens e ambientes, bem como as principais tecnologias a serem utilizadas. Nesta etapa, o gênero, público-alvo, plataformas-alvo e a jogabilidade em alto nível são assertados e colocados por escrito no *Game Design Document* (GDD). Produto: *Game Design Document*.
- **Determinar atores e estórias de usuário em alto nível**: a partir do documento do jogo, são definidos os atores que formam o jogo. As *features* identificadas no GDD são mapeadas em estórias de usuário em alto nível[1] [COHN, 2004]. Tais estórias de usuário compõe a primeira versão do *product backlog*. Produto: diagrama de casos de uso e *product backlog*.
- **Discussão sobre o jogo (*story-gathering workshop*)**: nesta etapa os times e os *stakeholders* se reúnem para discutir o jogo e os resultados dos testes com os usuários, se houver. Os presentes são incentivados a exporem suas ideias, dificuldades e sugestões. Os apontamentos podem refletir em alteração, criação ou exclusão de estórias de usuário. Em seguida, são selecionadas a(s) *epic*(s) de maior prioridade, que irão fazer parte do próximo *release*. Faz-se uma discussão das *epics* selecionadas, analisando-se detalhadamente os requisitos das mesmas. A medida que a discussão acontece, estas são divididas em estórias de usuário menores e mais detalhadas. Em geral, a discussão sobre o jogo ocorre no início do projeto e após a entrega dos *releases*. Produto: *product backlog* atualizado.
- **Elaboração das estórias de usuário**: Para cada estória de usuário discutida na etapa anterior, é feito a elaboração de cartões com esboço de interface, casos de teste e condições de satisfação, de modo a esclarecer os requisitos de forma detalhada. Se necessário, as estórias de usuário podem ser divididas em estórias de usuário menores. Produtos: esboço de interface, casos de teste e condições de satisfação.
- **Planejamento do *release* ou protótipo**: nessa etapa é definido o plano do próximo *release* ou protótipo. As estórias de usuário do *product backlog* são classificadas por prioridade e então é decidida a quatidade de *sprints* que irão fazer parte do *release*, bem como o objetivo de cada *sprint*, além do prazo final da entrega do *release*. Produto: plano do *release*.
- **Entrega do *release* ou protótipo**: essa etapa ocorre quando todos os *sprints* que compõe o *release* ou protótipo são finalizados ou quando o prazo para a entrega do *release* chega ao fim. Nessa etapa, o *release* é disponibilizado entre todos os envolvidos no projeto, sendo que estes são incentivados a testarem o jogo, especialmente com respeito a diversão e jogabilidade, a fim de fazerem novamente um *brainstorm* na etapa de discussão. Adicionalmente, alguns *releases* poderão ser distribuídos para a mídia especializada ou para o público

[1] Estórias de usuário em alto nível são chamadas de *epics*.

em geral na forma de *demos*, a fim de prospectar a aceitação do jogo pelo público-alvo. Produto: *feedback*.

- **Entregar a versão final**: após a discussão do último *release*, caso não hajam estórias de usuário pendentes, o jogo é comercializado, seja através de mídias ou pela Internet.

A **fase de execução**, onde os *sprints* são planejados em detalhes e então executados, é composta pelas seguintes etapas:

- **Definir e estimar tarefas**: essa etapa marca o início de cada *sprint*. As estórias de usuário com maior prioridade são selecionadas. Em seguida, define-se as tarefas necessárias para implementar a estória de usuário, incluindo esforços de programação, arte, som e testes. Estima-se a quantidade de esfoço para completar as tarefas. Caso as tarefas caibam no *sprint*, são alocadas no *sprint backlog*. Caso contrário, a estória de usuário é dividida ou postergada para o próximo *sprint*. No decorrer do *sprint* essa etapa pode ser revisitada para redefinir as tarefas, sem, no entanto, alterar quais são as estórias de usuário a serem implementadas. Produto: *product backlog* atualizado, *sprint backlog* e cartões de tarefas.
- **Reunião de pé**: cada dia é iniciado com uma reunião rápida e de pé, na qual todo o integrante da equipe relata o que fez no dia anterior e o que pretende realizar. Ao final, os participantes assumem as tarefas, realocando-as no quadro e atualizando o gráfico de *burndown*. Produtos: quadro de tarefas e gráfico *burndown*.
- **Sessão de modelagem ágil**: caso necessário, na sessão de modelagem ágil são modelados os diagramas e esboços para a execução das tarefas. Produtos: diagramas ou esboços de solução.
- **Sessão de implementação**: durante a implementação das tarefas são aplicadas as práticas de desenvolvimento guiado por teste, refatoração e integração contínua, tanto de código fonte quanto de arte, som ou outros artefatos do jogo. Produtos: testes unitários, código fonte, arte, som, etc.
- **Executar atividades de qualidade**: após a integração dos módulos, o resultado do *software* deve ser testado por uma equipe especializada, realizando-se testes caixa preta que atendam aos casos de teste definidos para as estórias; testes de cobertura; testes de integração; e, caso necessário, testes de carga [PRESSMAN, 2006]. Os defeitos e aperfeiçoamentos podem ser relatados em uma ferramenta de *bug tracking* e lançados como novas tarefas no próprio *sprint* ou no seguinte. Produto: Relatório de *bugs* e *postits*.
- **Integrar versão**: se todas as estórias implementadas atendem aos casos de teste ou atingiu-se o prazo final do *sprint*, o jogo é integrado em uma versão para demonstração. A versão resultante do primeiro *sprint* pode ser utilizada no *Greenlight Process*. Versões integradas em novos *sprint* podem ser rotulados, tradicionalmente, como Alfa ou Beta. Produto: versão do jogo.

- **Realizar testes com jogadores**: a versão corrente do jogo é testada por usuários finais, ou seja, jogadores com pouco ou nenhum conhecimento do jogo. Os *bugs* e problemas são documentados e utilizados como fonte de informação na sequência do projeto. Os testes com jogadores não são obrigatórios a cada *sprint*. Eles devem ser feitos periodicamente segundo um cronograma ou quando os tomadores de decisão acharem necessário. Produto: relatório de *bugs* atualizado.
- **Revisar o *sprint* (*sprint review*)**: o *product owner* e opcionalmente os *stakeholders* testam a versão corrente do jogo para validar ou não o trabalho desenvolvido no *sprint*. *Features* cuja implementação não atende aos requisitos de qualidade podem voltar ao *product backlog* ou serem canceladas. Produto: *product backlog* atualizado.
- **Reunião de retrospectiva**: ao final do *sprint* é realizada uma reunião, com toda a equipe, objetivando refletir sobre o processo de trabalho realizado, propondo aperfeiçoamentos para a próxima iteração. Essa etapa marca o fim do *sprint*. Produto: propostas de aperfeiçoamento do processo.

Na figura 3.4, as fases e etapas do GAMA estendido são apresentadas na forma de um diagrama de atividades. Em amarelo estão as principais modificações em relação ao GAMA original.

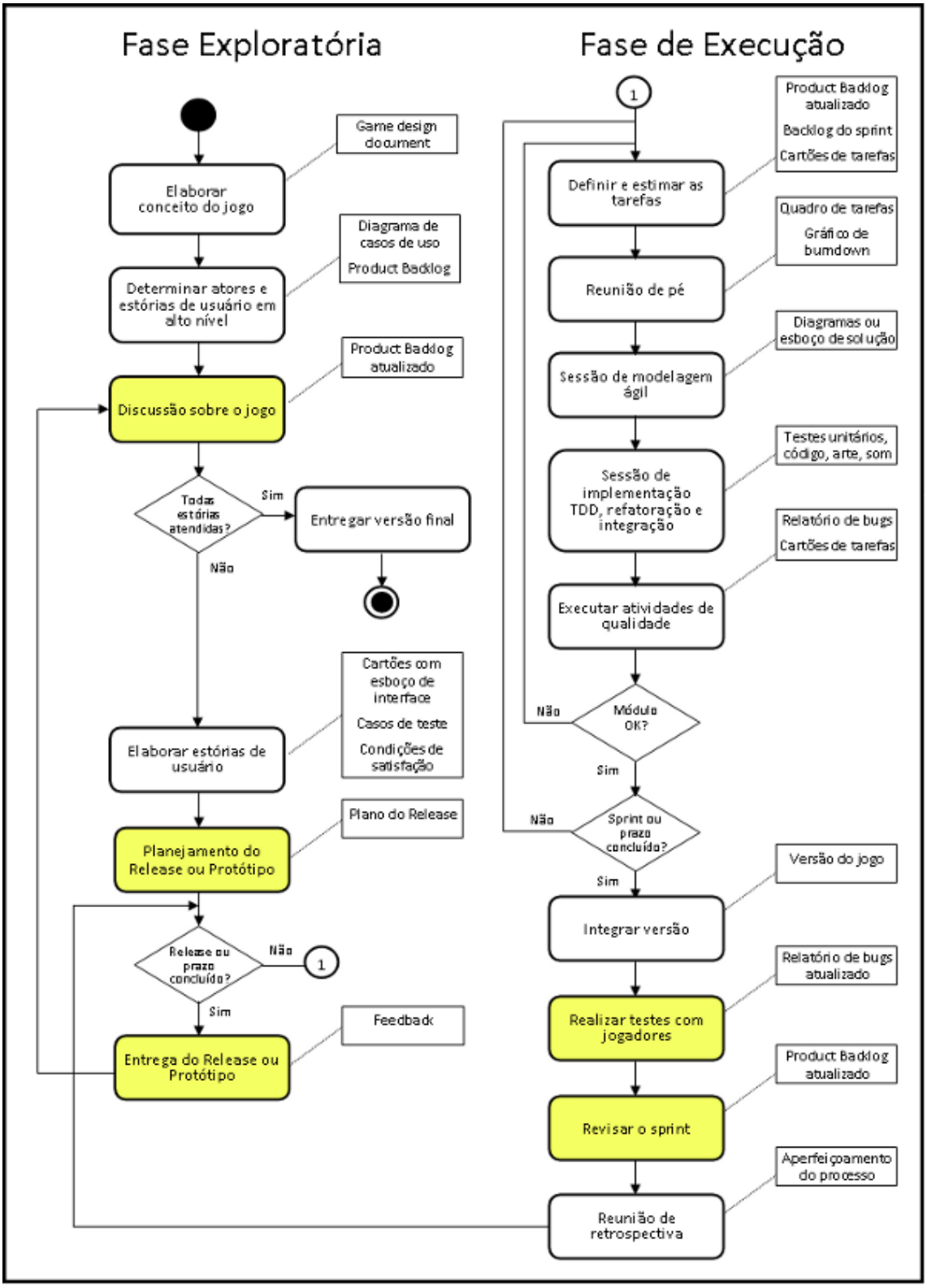


Figura 3.4 Diagrama de atividades do GAMA estendido

Conforme pode-se perceber, o GAMA foi modificado pontualmente, com o intuito de incorporar os aprimoramentos identificados na análise realizada e tornar a metodologia mais clara e apropriada para projetos de jogos eletrônicos. Entretanto, as principais ideias do GAMA original, como a forte aderência as práticas ágeis e a divisão em duas fases foram mantidas. O que segue são alguns comentários sobre as alterações propostas e sobre o ciclo de vida de um projeto que segue o GAMA estendido.

Em várias partes da metodologia nota-se a preocupação de procurar e explorar a diversão do jogo, bem como mitigar os detalhes apenas quando há informação necessária para isso. Após a definição do *Game Design Document*, as estórias de usuário em alto nível, as *epics*, são identificadas e discutidas. Depois dessa consideração inicial, na qual os fundamentos do conceito do jogo são definidos, cada *epic* é planejada e desenvolvida em um *release*. Caso o grau de incertezas sobre a *epic* em questão seja muito grande, pode-se planejar um protótipo para esclarecer as questões pendentes.

Cada *release* é composto por um conjunto de *epics* relacionadas entre si. As *epics* selecionadas no *release* definem o objetivo do mesmo. Por exemplo, num jogo do tipo *first-person shooter*, o objetivo de um *release* poderia ser desenvolver as armas que o jogador terá a disposição. Após a seleção das *epics* que irão compor o *release* e do objetivo do mesmo, é realizada uma divisão em *sprints*. No exemplo mencionado, poderia-se dividir a implementação de cada arma num *sprint*. Nesse momento, é definido o objetivo de cada *sprint*.

Durante o planejamento do *release* os detalhes vão naturalmente sendo discutidos e decisões são tomadas. No planejamento de cada *sprint*, estes detalhes são esclarecidos de forma ainda mais cabal. Neste sentido, a metodologia prevê um processo gradual de detalhamento: a medida que o desenvolvimento do *release* avança, o nível de detalhamento das *features* incorporadas pelo *release* gradativamente aumenta.

O nível de detalhamento gradual é ilustrado por [COHN, 2008] como um *iceberg*, conforme mostrado na figura 3.5. As estórias de usuário mais prioritárias são como as pequenas bolas de neve no topo do *iceberg*, sendo que estas são suficientemente detalhadas e pequenas para serem atacadas no *sprint* atual. Logo abaixo, estão os grandes blocos de gelo do *iceberg*, que representam as estórias de usuário menos prioritárias e, consequentemente, menos detalhadas que compõe o *release* atual. Por fim, temos a base do *iceberg*, abaixo da água, que são as *features* ainda não planejadas nem detalhadas a serem atacadas em futuros *releases*. Note que o nível de clareza, indicado pela cor dos blocos, aumenta na direção do topo do *iceberg*.

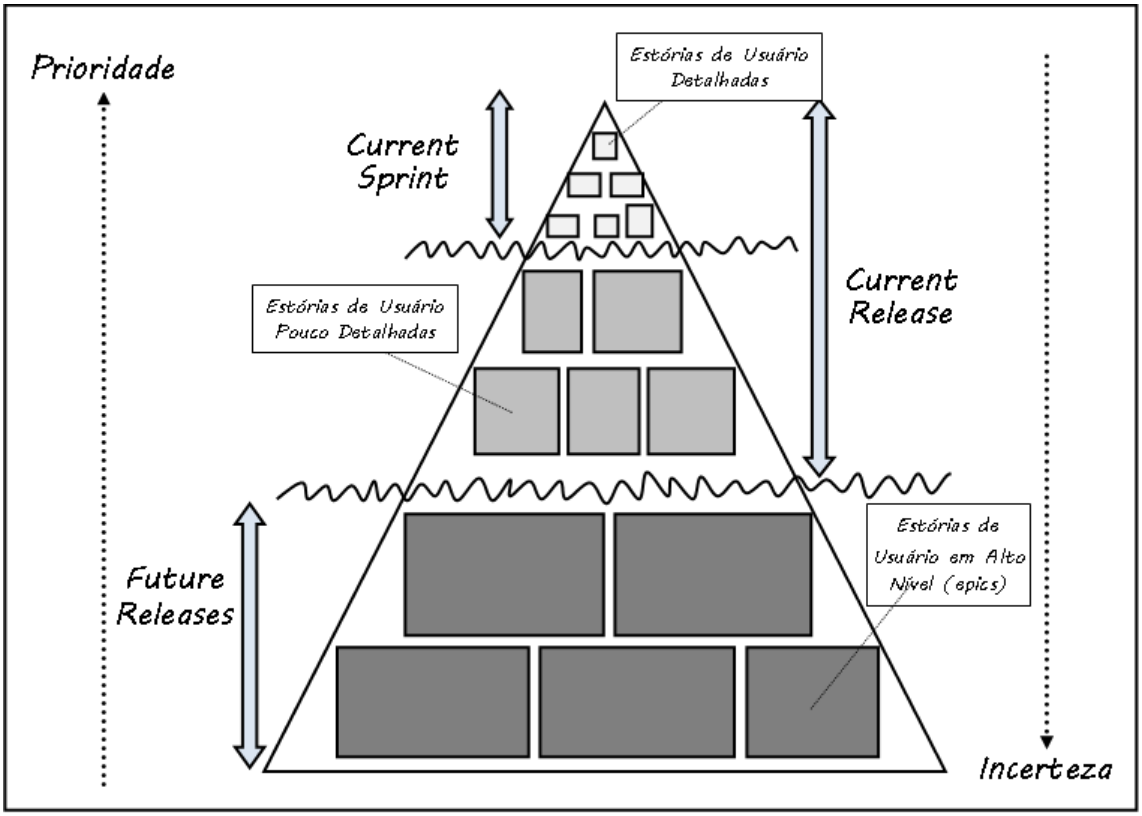


Figura 3.5. O *product-planning iceberg* [COHN, 2008; KEITH, 2010]

Segundo [KEITH, 2010], a postergação do detalhamento das *features* menos prioritárias, além de evitar o disperdício de esforço – visto que tais *features* tem grande probabilidade de serem alteradas até chegar o tempo de implementá-las – é importante para que o *product backlog* não fique tão grande que o torne difícil de visualizar e utilizar.

Um aspecto que é muito destacado no GAMA estendido é a preocupação de planejar e desenvolver primeiro o que é mais importante, característica herdada do *Scrum* [HIGHSMITH, 2002]. Esta preocupação reflete na constante priorização do *product backlog*, sendo que as estórias de usuário mais significativas são atacadas primeiro. A priorização do *product backlog* é feita em diversos momentos diferentes. Na etapa de discussão do jogo, as estórias de usuário de alto nível são priorizadas; esta priorização tem um grande impacto sobre a sequência do projeto. Na etapa de planejamento do *release*, apenas as estórias de usuário que compõe o *release* são priorizadas; esta priorização impacta somente o desenvolvimento do próprio *release*. Na etapa de definição e estimativa de tarefas as estórias de usuário que compõe o *sprint* são priorizadas, com impacto somente no próprio *sprint*. A priorização do *product backlog* nunca é definitiva. A revisão das prioridades das *features* que compõe o *product backlog* com base no conhecimento e experiência adquiridos no decorrer do projeto é um dos principais aspectos da adaptabilidade prevista no GAMA estendido.

Somando-se a forma gradual de detalhar as *features* do jogo, tem-se o *feedback* gerado através dos vários testes previstos na metodologia. A cada fim de *sprint*, a

equipe de desenvolvimento e os *designers* testam o jogo, tanto para avaliar a diversão quanto para encontrar *bugs*. Além disso, periodicamente são executados testes com jogadores, ou sempre que for julgado necessário. Adicionalmente, no final de cada *release* (ou protótipo), que usualmente são marcos importantes no progresso do jogo, o mesmo é testado pelos *stakeholders*.

Uma diferença importante entre o GAMA original e o GAMA estendido é que neste último explicita-se que o conceito do jogo pode ser modificado (sempre de forma controlada) durante o decorrer do projeto. No GAMA original, a fase exploratória é revisitada apenas para fazer o planejamento das iterações, ou *sprints*, sendo que a definição e elaboração das estórias de usuário são realizadas apenas no início do projeto. Em contraste, no GAMA estendido incentiva-se a discussão do jogo a cada fim de *release*, a fim de rever o conceito do mesmo e (re)elaborar ou repriorizar as estórias de usuário. Neste sentido, no GAMA estendido a agilidade é utilizada de forma mais direta não apenas no planejamento e execução dos *sprints*, mas também na definição do conceito do jogo.

A constante (re)priorização do *product backlog*, a forma gradativa de detalhar as estórias de usuário e a possibilidade de ajustar o conceito do jogo durante o progresso do jogo pode ser constatada de forma bastante clara no fato de que várias etapas do diagrama de atividades do GAMA estendido terem como artefato produzido o *product backlog* atualizado.

A etapa de discussão do jogo, chamada de *story-gathering workshop* por [KEITH, 2010], é uma das etapas mais importantes do GAMA estendido. Sob muitos aspectos, é principalmente nessa etapa que o conhecimento e experiência agregados durante o decorrer do projeto é organizado, mitigado e utilizado. O *feedback* gerado por todos os testes, bem como as conclusões provenientes de eventuais protótipos são a base da discussão e das consequentes alterações ou ajustes que serão realizados na sequência do projeto. Durante a etapa de discussão do jogo os três principais aprimoramentos identificados na análise crítica convergem com o objetivo de permitir a adaptação natural do jogo com base nas informações coletadas até o momento, sempre visando encontrar e explorar da melhor forma possível a diversão do jogo.

Por fim, apresentamos abaixo o diagrama de atividades do GAMA original e do GAMA estendido lado a lado, para melhor visualização e comparação das mudanças propostas.

GAMA original

GAMA estendido

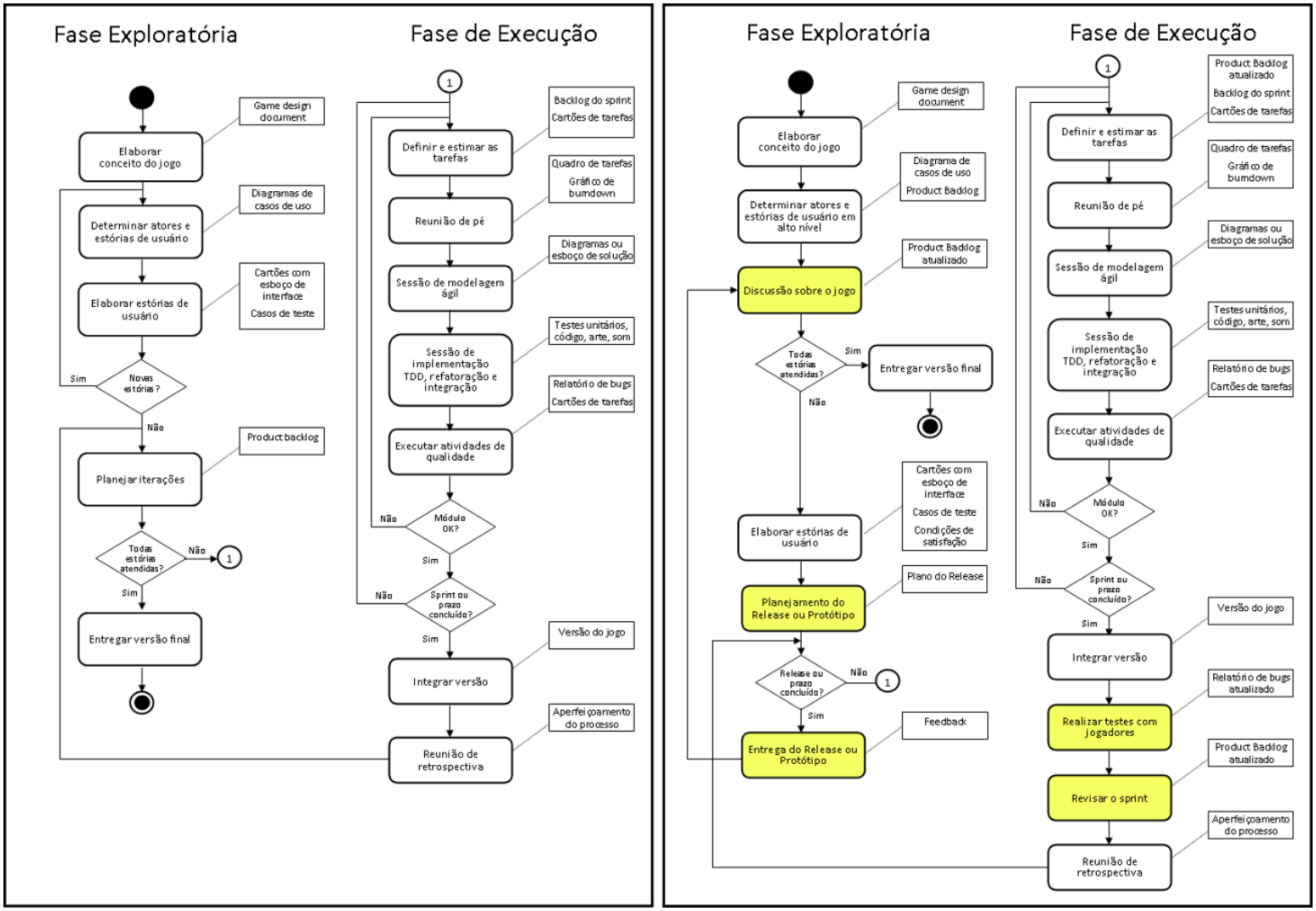


Figura 3.6. GAMA original e estendido

4. ESTUDO DE CASO: APLICANDO PRÁTICAS ÁGEIS

No capítulo anterior, realizamos uma análise sobre o *Game Agile Methods Applied* (GAMA) e consideramos argumentos que justificam algumas mudanças pontuais no mesmo. Na sequência, apresentamos a versão modificada do GAMA, que incorpora os aprimoramentos previamente identificados, além de outras mudanças que visam deixá-lo mais adequado as exigências e complexidades de desenvolver jogos.

Após a elaboração da versão estendida do GAMA, algumas práticas ágeis que o compõe foram aplicadas num estudo de caso. O objetivo deste capítulo é relatar como e porque estas práticas mostraram ser valiosas durante o desenvolvimento do experimento. O *Game Design Document* do jogo desenvolvido, bem como alguns dos artefatos produzidos durante o desenvolvimento são encontrados como Apêndices.

4.1 Definindo o escopo do estudo de caso

Idealmente, gostaríamos de aplicar integralmente o GAMA no desenvolvimento de um jogo para conseguir validá-lo de forma mais completa, respeitando o método científico. Isso envolve a colaboração de várias pessoas, sendo que aplicar o GAMA num ambiente de desenvolvimento real seria extremamente valioso. Infelizmente, não conseguimos alcançar este objetivo.

Por esta razão, o estudo de caso teve o escopo limitado a uma pessoa desenvolvendo um jogo simples durante um período de aproximadamente dois meses. Consequentemente, não foi possível aplicar o GAMA de forma completa. Várias etapas e atividades, como as reuniões e discussões, testes com jogadores e estimação das tarefas utilizando a técnica *Planning Poker* exigem a participação de uma equipe completa de desenvolvimento. Mais comentários sobre as limitações do estudo de caso e as perspectivas de trabalhos futuros relacionadas são descritas na conclusão deste trabalho. A figura 4.1 ilustra o escopo do estudo de caso.

4.2 Práticas ágeis aplicadas no estudo de caso

Considerando o escopo do estudo de caso, algumas das práticas ágeis, atividades e características que compõe o GAMA e puderam ser total ou parcialmente aplicadas foram:

- Simplicidade
- Priorização regular das *features*
- Utilização de *Product Backlog* e estórias de usuários

- Detalhamento gradual das *features*
- Iterações curtas

Nas seções que segum, iremos considerar como cada item acima mostrou ser de valor durante o desenvolvimento do estudo de caso.

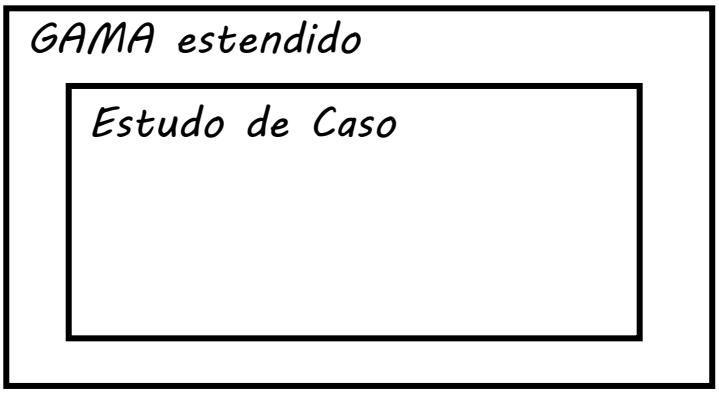


Figura 4.1. Escopo do estudo de caso

4.2.1 Simplicidade

A simplicidade é um valor comum entre as práticas ágeis, sendo um dos pilares do *Extreme Programming* (XP) [BECK, 1999] e da Modelagem Ágil (MA) [AMBLER, 2004]. Por consequência, a simplicidade também faz parte do GAMA [PETRILLO, 2008].

No estudo de caso, a simplicidade foi aplicada principalmente com respeito a documentação elaborada no início do projeto. Isso é corroborado por um dos valores descritos no Manifesto Ágil [AGILE ALLIANCE, 2006]:

> *"Nós valorizamos software em funcionamento mais que documentação abrangente."*

Naturalmente, isso não significa que as práticas ágeis desincentivam toda e qualquer documentação. A ideia principal é que a documentação ágil deve ser o mais simples possível, tão mínima quanto possível e com um propósito bem definido [AMBLER, 2004]. Outro aspecto destacado nos projetos ágeis é que mesmo que documentação e modelos sejam importantes, nunca se deve esquecer que entregar o *software* é a prioridade [AGILE ALLIANCE, 2006; COCKBURN, 2000].

No contexto do XP e da MA, uma das características da simplicidade é não antecipar requisitos futuros [HIGHSMITH, 2002]. É melhor fazer coisas simples hoje e investir um pouco mais amanhã do que construir e projetar coisas mais complicadas hoje que talvez jamais sejam usadas [BECK, 1999]. O fundamental é documentar, planejar e modelar apenas o que é necessário para satisfazer as necessidades do presente [FOWLER, 2001]. Evitar prever requisitos futuros é extremamente importante em projetos de desenvolvimento de jogos, devido as grandes incertezas inerentes [KEITH, 2010].

Aplicando a simplicidade no estudo de caso, foi elaborada uma documentação objetiva e mínima na forma do *Game Design Document* e diagramas de casos de uso. Assim, temos a seguinte definição propositalmente concisa do jogo que foi desenvolvido no experimento:

> *"Spyder's Saga é um jogo para PC do tipo platformer. O jogador utiliza o teclado para controlar o personagem principal, a aranha. Nas fases, o jogador terá de eliminar inimigos, resolver pequenos puzzles e utilizar as teias da aranha para avançar até o final."*
>
> *Game Design Document* do jogo Spyder's Saga

Como consequência da documentação simples, as decisões que não eram fundamentais no presente foram postergadas. Conforme apoiado pelo princípio do Desenvolvimento Enxuto de *Software*, **decida o mais tarde possível**, num ambiente onde as incertezas são grandes, deixar as decisões em aberto é mais interessante do que tomá-las cedo demais [POPPENDIECK, POPPENDIECK, 2003]. As decisões devem ser feitas baseados em fatos, não em especulações [PETRILLO, 2008]. Assim, muitos detalhes do jogo foram propositalmente desconsiderados no início do projeto, sendo que os mesmos devem receber maior atenção a medida que o projeto avança.

A simplicidade aplicada desde as primeras etapas do projeto permitiu que fosse gasto mais tempo desenvolvendo o jogo propriamente dito. Isso foi importante tendo em vista o objetivo do primeiro *release*:

> *"Objetivo do Release: desenvolver o mais rapidamente possível uma versão jogável que permita testar as principais features."*
>
> Plano do *Release* 1

Portanto, foi definido que durante o primeiro *release* uma versão inicial das principais *features* do jogo seriam implementadas para testar o conceito do jogo o mais rapidamente possível e então começar a busca pela diversão imediatamente. Naturalmente, foi implementada uma versão simplificada das *features*. Dessa forma, a simplicidade incentivada pelas práticas ágeis ajudou a focalizar no que era mais importante no momento: uma versão do jogo funcionando, conforme apoiado pelo seguinte princípio expresso no Manifesto Ágil [AGILE ALLIANCE, 2006]:

> *"Ter o software funcionando é a principal medida de progresso."*

Como comparação, dificilmente a mesma estratégia poderia ser aplicada caso fosse utilizado o processo em cascata. Por exemplo, se o processo em cascata fosse aplicado integralmente, os menus, a história do jogo, bem como os *assets* teriam de ser documentados e projetados, mesmo que inicialmente fosse de forma simplificada. Entretanto, tal documentação adicional não teria nenhum valor no presente, e seria potencialmente prejudicial, visto que possivelmente essa documentação teria de ser refeita ao longo do projeto [KEITH, 2010]. Adicionalmente, isto retardaria o desenvolvimento da primeira versão do jogo e, consequentemente, postergaria atividades importantes como testes e validação do conceito do jogo.

4.2.2 Priorização regular das *features*

Um jogo real pode ter dezenas de *features* e milhares de *assets* a serem projetados e desenvolvidos. Cada uma destas *features* ou *assets* pode requerer várias tarefas para ficarem prontos. Keith [2010] considera que isso é tão crítico que, em projetos que utilizam o *Scrum*, o grande volume de itens contidos no *Product Backlog* pode ser um problema. Diante de tantas coisas a desenvolver, se focar primeiro no que é mais importante é extremamente útil.

Para se focar sempre no que é mais importante, o *Scrum* incentiva a constante priorização das tarefas, *features*, ou estórias de usuário [HIGHSMITH, 2002]. No contexto de desenvolvimento de jogos, ao invés de utilizar a abordagem "implementar tudo que está no GDD", as *features* são desenvolvidas seguindo uma ordem definida com base no valor que representam para o jogador que irá comprar o jogo [KEITH, 2010]. Aquilo que é mais prioritário recebe maior atenção; o que é menos prioritário é considerado mais tarde. Um dos benefícios de utilizar a priorização regular dos requisitos é que caso seja necessário diminuir o escopo do jogo, é relativamente simples identificar quais *features* são menos valiosas para o jogador, e portanto, potenciais candidatas a serem descartadas [KEITH, 2010] .

Visto que o GAMA utiliza o gerenciamento de projetos baseado no *Scrum*, a priorização regular das *features* é uma das atividades que fazem parte da metodologia. No GAMA estendido, essa atividade foi ainda mais destacada, sendo que é incentivada a priorização das *features* em vários etapas distintas do projeto. No estudo de caso, a priorização foi realizada em três momentos, conforme previstos implicita ou explicitamente no GAMA estendido:

- **Priorização no Planejamento do *Release***: entre as muitas *features* que fazem parte do jogo e que ainda não foram desenvolvidas, selecionou-se um sub-conjunto para ser atacado durante o *release*. Estas *features* foram selecionadas levando em conta o valor que elas agregam ao jogo. Quanto maior o valor de uma *feature* sob o ponto de vista o jogador, maior a sua prioridade. Assim, as *features* não selecionadas são implicitamente menos prioritárias.
- **Priorização entre *Sprints***: durante o planejamento do *release*, as *features* selecionadas são distribuídas entre um conjunto de *sprints* que compõe o *release*. Após essa atividade, a ordem de execução dos *sprints* foi definida segundo sua prioridade em relação aos demais.
- **Priorização no início de cada *Sprint***: após a definição das tarefas necessárias para desenvolver cada *feature* que compõe o *sprint*, as mesmas são ordenadas por prioridade. Tarefas mais importantes do ponto de vista do jogador tem maior prioridade e, portanto, são as primeiras a serem implementadas.

As priorizações acima mostraram-se apropriadas, visto que ajudaram a manter o foco naquilo que agrega maior valor ao jogo e fez o desenvolvedor não esquecer de ver o jogo sob o ponto de vista de quem mais importa, o jogador [GIBSON, 2007]. Embutidas as priorizações está uma reflexão do jogo e do processo. Durante as atividades de priorização, é comum perguntar-se: quais dos requisitos levantados é mais importante para o jogador? Será que há algum requisito importante para o

jogador que não identifiquei? A última priorização realizada mostrou-se correta? Que impacto haverá caso haja necessidade de mudanças nas prioridades? A priorização está realmente sendo feita sob o ponto de vista do jogador? Ou será que o olhar de desenvolvedor está influenciando nas escolhas? Tais perguntas pertinentes colaboraram para o aperfeiçoamento do jogo e do processo.

As tabelas 4.1 e 4.2 mostram o *product backlog* antes e depois da primeira priorização. Esta priorização foi bastante simples, pois foi realizada sobre estórias de usuário de alto nível no início do planejamento do *release*, quando decidiu-se quais *features* seriam atacadas. Observe que os elementos essenciais para o desenvolvimento da primeira versão jogável do jogo (cenário, câmera, controles, IA) foram classificados com prioridade maior do que os demais (som, física, história, etc). Outro detalhe é que, no momento que a priorização foi realizada, a única informação relevante era saber que as *features* selecionadas no *release* possuíam prioridade maior do que as demais.

Epic N°	**Epic**	**Priority**
1	Story	-
2	Tutorial	-
3	Sounds	-
4	Scenario	-
5	IA	-
6	Camera Control	-
7	Character Control - Physics	-
8	Character Control - Input	-

Tabela 4.1. *Product Backlog* antes da primeira priorização.

Epic N°	Epic	Priority
4	Scenario	HIGH
5	IA	HIGH
6	Camera Control	HIGH
8	Character Control - Input	HIGH
1	Story	LOW
2	Tutorial	LOW
3	Sounds	LOW
7	Character Control - Physics	LOW

Tabela 4.2. *Product Backlog* após a primeira priorização.

4.2.3 Utilização do *Product Backlog* e estórias de usuário

O *Product Backlog*, artefato fundamental utilizado no *Scrum*, é definido como uma "Lista de requisitos ou *features*" que são priorizados e gradualmente refinados [SCRUM ALLIANCE, 2011]. Uma característica importante do *Product Backlog* é que ele é dinâmico: novos requisitos são adicionados, enquanto que requisitos completados são removidos [KEITH, 2010; SCHWABER, 2004].

No estudo de caso, conforme definido no GAMA , os requisitos e *features* do jogo foram mapeados em forma de estórias de usuário. Segundo Keith [2010], algumas características das estórias de usuário que as fazem serem apropriadas para o desenvolvimento de jogos são:

- Comunicam a prioridade e o valor das *features*
- Permitem que os detalhes sejam identificados a medida que mais informação é aprendida

Para expressar as estórias de usuário, foi utilizado o template recomendado por Cohn [2004]:

*Como um <**papel de usuário**> , eu quero <**objetivo**> [por esta razão <**motivo**>]*

A tabela 4.3 mostra o *Product Backlog* em determinado momento do desenvolvimento do estudo de caso.

Product Backlog
CHARACTER CONTROL
Movimentação
Como um jogador, quero que a aranha se movimente nem rápido nem devagar demais.
Como um jogador, quero que a aranha sempre esteja virada na direção do último movimento.
Como um jogador, quero controlar a movimentacao da aranha
Como um jogador, quero saber quando a aranha está se movimentando.
Como um jogador, quero que a aranha caia se não estiver em solo firme.
Pulo
Como um jogador, quero saber quando a aranha está executando um pulo.
Como um jogador, quero controlar o pulo da aranha.
Como um jogador, quero que o pulo esteja desabilitado caso a aranha nao esteja em solo firme.
WEB
Como um jogador, quero saber quando o tempo da plataforma temporária está se esgotando.
Como um jogador, quero que a teia de ataque não tenha efeito caso saia do campo de visão.
Como um jogador, quero que a teia de ataque seja destruida caso se choque com algum objeto.
Como um jogador, quero controlar quando a teia de ataque se transforma em uma plataforma temporária.
Como um jogador, quero saber quando a teia de ataque se transformou em plataforma temporária.
Como um jogador, quero controlar quando a teia de ataque é lançada.
Como um jogador, quero saber quando a teia de ataque atingiu ou não o inimigo.
Como um jogador, não quero que uma teia se sobreponha a outra.
Como um jogador, quero que o uso da plataforma teporária seja desafiador e demande agilidade.
Como um jogador, quero poder customizar os controles.
Como um jogador, quero que os controles default sejam teclas frequentemente utilizadas por jogos do mesmo gênero.
MISSIONS
Story
Tutorial
Sounds
...

Tabela 4.3. *Product Backlog*

A utilização do *Product Backlog* em conjunto com estórias de usuário mostrou ser bastante aproriada, pois permitem expressar os requisitos do jogo sem preocupação excessiva com detalhes técnicos. Os detalhes técnicos são abordados principalmente durante a elaboração das tarefas dos *sprints*, o que permite que as estórias de usuário sejam concisas e objetivas. Dessa forma, as estórias de usuários são muito comunicativas e fáceis de serem utilizadas. Adicionalmente, o uso do *Product Backlog* composto por estórias de usuário facilita atividades importantes tais como priorização, estimação de esforço e detalhamento gradual das *features* [KEITH, 2010].

4.2.4 Detalhamento gradual das *features*

Como já mencionado, uma das características de projetos ágeis é projetar e modelar apenas o que é necessário no presente [BECK, 1999; HIGHSMITH, 2002]. No estudo de caso, isso refletiu numa documentação tão simples quanto possível que atendesse as necessidades imediatas [AMBLER, 2004; FOWLER, 2001].

Contudo, a elaboração do conceito do jogo na forma do *Game Design Document*, diagramas de casos de uso e a identificação das primeiras estórias de usuário, chamadas de *epics*, não é suficientemente claro e detalhado para ser implementado, testado e validado. Esta simplicidade proposital é compensada por um detalhamento gradual realizado antes de passar para a fase de execução.

Visto que detalhar todas as *features* no início do projeto pode levar vários dias para ser realizado, é potencialmente um desperdício de recursos e pode tornar o *Product Backlog* ingerenciável, o *Scrum* incentiva que o detalhamento deve ser realizado somente sobre os requisitos ou *features* com maior prioridade que serão implementados no(s) próximo(s) *sprint*(s) [KEITH, 2010].

Aplicando esta abordagem prevista no GAMA no estudo de caso, foram realizados a seguinte sequência atividades:

1. As estórias de usuário de alto nível (*epics*) mais importantes foram selecionadas. Tais estórias de usuário compõe o primeiro *release*.
2. As estórias de usuário selecionadas no passo anterior foram “quebradas” em estórias de usuário menores e mais claras, revelando detalhes até então não identificados ou mencionados.
3. Após a distribuição das estórias de usuário detalhadas entre *sprints*, as mesmas são novamente “quebradas” em estórias de usuário menores, se necessário.

Ao elaborar as estórias de usuário que serão utilizadas na fase de execução, é necessário seguir algumas diretrizes:

- As estórias de usuário devem ser suficientemente claras de modo que o desenvolvedor ou artista saiba exatamente o que deve fazer
- As estórias de usuário devem ser suficientemente pequenas de modo a “caber” dentro de um *sprint*
- As estórias de usuário devem ser facilmente priorizadas, testadas e validadas

A figura 4.2 mostra um exemplo do detalhamento gradual das *features* realizado no estudo de caso.

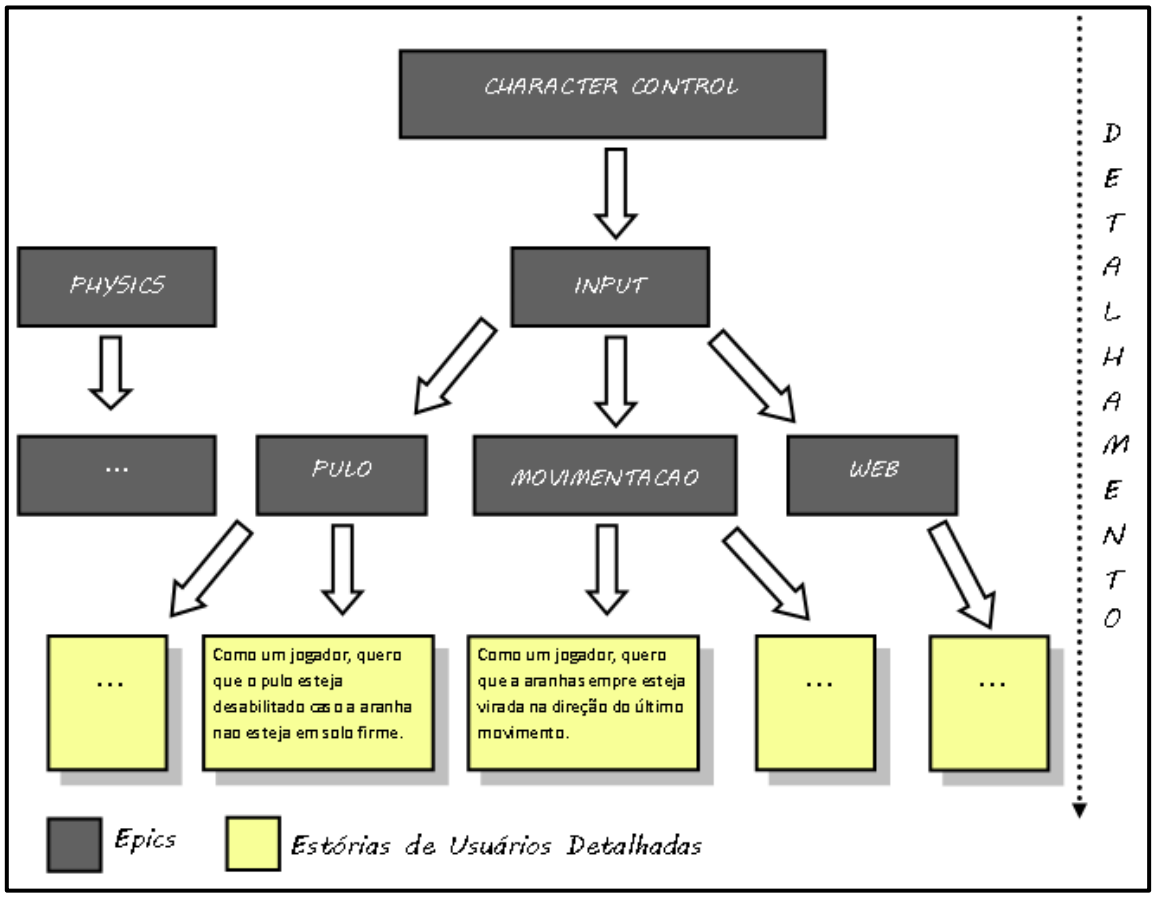


Figura 4.2. Detalhamento gradual das *features*.

O detalhamento gradual realizado no estudo de caso mostrou ser apropriado e valioso. Com essa abordagem, as *features* são mitigadas apenas no momento que é necessário. Inicialmente, as *features* selecionadas para compor o *release* corrente são detalhadas até que fique claro o que precisa ser feito. Se necessário, no início de cada *sprint*, as estórias de usuário que o compõe são novamente "quebradas" e detalhadas. Resumidamente, o detalhamento gradual permite discutir, entender e visualizar o que é necessário no momento que é necessário, ao passo que as *features* não selecionadas permanecem intactas.

4.2.5 Iterações curtas

Iterações curtas são uma característica fundamental dos projetos ágeis, conforme apoiado pelo princípio expresso no Manifesto Ágil [AGILE ALLIANCE, 2006]:

> *"Entregar softwares em funcionamento com frequência de algumas semanas a alguns meses, de preferência na menor escala de tempo."*

Este princípio é aplicado com destaque no *Scrum*, visto que iterações curtas (*sprints*) de duas a quatro semanas são uma das principais características do processo [HIGHSMITH, 2002]. No GAMA estendido, iterações mais longas (*releases*) são compostos por iterações curtas (*sprints*). Entre cada *release* espera-se que um conjunto de *features* relacionadas entre si sejam desenvolvidas, resultando numa

versão do jogo que seja uma forte candidata a ser utilizada como base de discussões e testes. Nos *sprints*, o objetivo é desenvolver uma *feature* ou um conjunto de *features* que seja(m) importante(s) para atingir o objetivo do *release*. Por exemplo, se o objetivo do *release* é "Desenvolver a Inteligência Artificial", o objetivo de um *sprint* poderia ser "Desenvolver o Algoritmo de Pathfinding".

Apesar de o estudo de caso não ter sido longo o suficiente para aplicar uma iteração longa (*release*) do início ao fim, pôde-se aplicar duas iterações curtas (*sprints*) de três semanas. Essa pequena amostra revelou que as iterações curtas ajudam a manter uma alta produtividade sem perder tempo com coisas sem valor ou que podem ser realizadas mais tarde.

As duas iterações curtas realizadas tiveram, respectivamente, os seguintes objetivos:

- **Objetivo do *Sprint* 1**:Desenvolver a primeira versão do cenário do jogo, composto por terreno, modelos e texturas. O cenário deve ser simplificado, porém condizente com a proposta do jogo (estilo cartoon).
- **Objetivo do *Sprint* 2**: Desenvolver os aspectos principais referentes a interação do jogador com o personagem principal

No estudo de caso, as iterações curtas auxiliaram a distribuir o tempo disponível de forma que maximisasse o valor do resultado final, atingindo o objetivo do *release* de forma mais rápida. Por exemplo, caso não fosse adotada a utilização de iterações curtas, poderia-se gastar muito tempo desenvolvendo o cenário (texturas, modelos, shaders, etc). Consequentemente, ter-se-ia um cenário melhor acabado, mas, em contrapartida, outras partes importantes do jogo ficariam sem ser desenvolvidas. Tendo em vista o objetivo do *release* (testar o mais rapidamente possível as principais *features*), iterações curtas em que são desenvolvidas diferentes *features*, mesmo que de maneira simplificada, são mais apropriadas do que iterações longas com enfoque total em um único aspecto do jogo. As figuras 4.3 e 4.4 mostram o progresso do jogo ao fim de cada *sprint*.

Figura 4.3. *Screenshot* do jogo ao fim do primeiro *sprint*.

Figura 4.4. *Screenshot* do jogo ao fim do segundo *sprint*.

5. CONCLUSÃO

Este capítulo finaliza o trabalho apresentando algumas conclusões. Primeiro, é apresentado o resumo dos resultados produzidos. Na seção seguinte, apresentamos as principais limitações enfrentadas ao longo do trabalho. Na última seção, são propostas algumas ideias e atividades que poderão guiar trabalhos futuros.

5.1 Resumo dos Resultados

Ao longo deste trabalho, produzimos alguns resultados condizentes com o objetivo do mesmo. Listamos abaixo os principais resultados obtidos:

1. Com base em várias referências, foram abordados alguns problemas do modelo em cascata quando aplicado ao desenvolvimento de jogos eletrônicos
2. Apresentamos motivos que justificam a utilização das práticas ágeis no desenvolvimento de jogos. A argumentação foi baseada em trabalhos que indicam que as práticas ágeis podem minimizar os problemas frequentemente encontrados na indústria de jogos, bem como casos de sucesso
3. Abordamos quatro metodologias ágeis ou parcialmente ágeis aplicadas ao desenvolvimento de jogos. Entre elas, apresentamos o GAMA, cujo estudo foi o foco principal deste trabalho
4. Realizamos uma análise sobre o GAMA. Como resultado, identificamos três aspectos não previstos no GAMA original (testes com jogadores) ou previstos de maneira limitada (prototipação e adaptabilidade). A importância destas atividades e práticas no desenvolvimento de jogos foi justificada utilizando referências da área. Adicionalmente, aprensentamos aspectos dos processos e metodologias ágeis que apóiam a utilização da prototipação, adaptabilidade e testes com jogadores
5. Elaborou-se uma versão estendida do GAMA. Com base na análise realizada e nos aprimoramentos identificados, foram propostas algumas mudanças no GAMA para torná-lo mais completo e eficiente em relação as necessidades do desenvolvimento de jogos eletrônicos. O foco principal foi incorporar a prototipação, testes com jogadores e maior suporte a adaptabilidade na metodologia. Além disso, foram realizadas outras alterações a fim de tornar o GAMA mais prático. Como resultado final, criou-se uma versão estendida do GAMA, composta basicamente por algumas etapas novas em adição as etapas do GAMA original.

6. Algumas práticas, valores, princípios e características ágeis que compõe o GAMA estendido foram aplicadas num estudo de caso. Os resultados alcançados por cada aspecto ágil aplicado foram:

   - **Simplicidade**: auxiliou na elaboração de uma documentação simples que atendesse as necessidades do presente, sem antecipar requisitos futuros. Isso, por consequência, permitiu iniciar o desenvolvimento das *features* do jogo tão cedo quanto possível.
   - **Priorização regular das *features***: incentivou a manter o foco naquilo que é mais importante para o jogador, além de servir como uma reflexão do jogo e do processo.
   - **Utilização do *Product Backlog* e estórias de usuário**: permitiu expressar os requisitos de forma simples e comunicativa, sem se preocupar excessivamente com detalhes técnicos, além de ser compatível com atividades como priorização e detalhamento gradual das *features*.
   - **Detalhamento gradual das *features***: permitiu mitigar os detalhes no momento apropriado, sem tornar o *Product Backlog* ingerenciável.
   - **Iterações curtas**: teve importância em limitar o tempo gasto no desenvolvimento de cada *feature*, evitando o enfoque total num aspecto do jogo em detrimento de outros igualmente importantes.

5.2 Limitações do Trabalho

Infelizmente, o trabalho sofreu de algumas limitações importantes devidos a fatores externos e simplificações necessárias. Abaixo são apresentadas as duas principais limitações enfrentadas ao longo do trabalho.

Limitações relacionadas as referências

A primeira grande limitação do trabalho tem relação com a coleta de referências sobre a utilização de métodos ágeis no desenvolvimento de jogos eletrônicos. Infelizmente tais referências não são encontradas com facilidade e em abundância. Obras como a de Keith [2010] são raras. Essa dificuldade é ainda maior em relação a casos reais documentados nos quais obteve-se ou não sucesso em aplicar métodos ágeis no desenvolvimento de jogos. Além disso, algumas referências, como *Game Unified Process* [FLOOD, 2003] e *Extreme Game Development* [DEMACHY, 2003], não são suficientemente claras e detalhadas. Essa escassez de referências torna difícil comparar o GAMA com outras metodologias ágeis aplicadas em. Naturalmente, esta limitação também é uma forte motivação para continuar a tentar esclarecer como a agilidade pode ser aplicada no mercado de jogos, incluindo os benefícios e riscos envolvidos.

Limitações do estudo de caso

A segunda grande limitação do trabalho está relacionada com o estudo de caso. Inicialmente, havia a intenção de desenvolver o estudo de caso utilizando o GAMA num ambiente de desenvolvimento real (uma pequena empresa de jogos). Nesse contexto, o objetivo do estudo de caso seria de testar, validar e consolidar o GAMA.

Infelizmente isso se tornou inviável por vários motivos. Com isso, o estudo de caso ficou restringido a um escopo muito simplificado, com várias limitações:

- apenas um desenvolvedor
- menos de dois meses de desenvolvimento
- dedicação parcial
- jogo muito simples

Diante das limitações acima, várias etapas do GAMA não puderam ser aplicadas ou foram aplicadas de forma muito simplificada. Por exemplo:

- a técnica chamada *Planning Poker* [KNIBERG,2007], utilizada para estimar o esforço envolvido no desenvolvimento das tarefas, não foi executada, visto que demanda um grupo de desenvolvedores
- os testes com jogadore não foi realizado, visto que o jogo não estava suficientemente maduro para isto
- o suporte a adaptabilidade não pôde ser avaliado, visto que a necessidade de adaptação ocorre principalmente em projetos maiores e após os primeiros meses de trabalho

Consequentemente, o estudo de caso não foi suficientemente profundo e completo para testar, validar e consolidar o GAMA. Tendo isso em mente, o escopo do estudo de caso foi simplificado, e apenas algumas características do GAMA puderam ser parcialmente aplicadas.

5.3 Perspectivas de trabalhos futuros

As limitações referentes ao estudo de caso deixam em aberto a necessidade de aplicar o GAMA estendido no desenvolvimento de um jogo real do ínicio ao fim, com um time de pesssoas envolvidas e recursos apropriados. Idealmente, esta aplicação deveria ser feita não no escopo acadêmico, mas em empresas profissionais de desenvolvimento de jogos.

Caso esse cenário ideal seja impossível de ser obtido, o GAMA poderia ser aplicado de forma mais completa do que a realizada neste trabalho, mesmo no escopo acadêmico. Por exemplo, poderia-se utilizar disciplinas como a de Jogos Eletrônicos do Instituto de Informática da Universidade Federal do Rio Grande do Sul, com os alunos utilizando o GAMA durante o projeto de quatro meses da diciplina (desenvolvimento de um jogo). Isso permitiria avaliar o GAMA observando mais rigorosamente o método científico.

Em suma, a perspectiva de trabalho futuro está relacionada a testar, validar e consolidar o GAMA, algo que não conseguiu-se realizar neste trabalho nem no trabalho que o originou.

Finalmente, numa visão mais abstrata, a limitação relacionada com a escassez de referências é uma grande motivação para que o trabalho futuro se concentre em contribuir, de alguma forma, a desvendar como a agilidade pode ser útil neste

fascinante, desafiador e frequentemente obscuro universo que é o desenvolvimento de jogos.

REFERÊNCIAS

AGILE ALLIANCE. **Principles: the agile alliance**. USA, april 2006. Disponível em: <http://www.agilemanifesto.org/principles.html>. Acesso em: junho de 2011.

AMBLER, S. **Modelagem Ágil**. São Paulo: Bookman, 2004.

BECK, K. **Extreme Programming Explained: embrace change**. 1ª ed. USA: Addison-Wesley Professional, 1999

BOEHM, B. **Software Engineering Economics**. Englewood Cliffs, NJ: Prentice Hall, 1981.

BETHKE, E. **Game Development And Production**. 1ª ed. Plano: Wordware Publishing, Inc, 2003.

BLOW, J. **Game development: Harder than you think**. Queue, ACM, New York, NY, USA, v. 1, n. 10, p. 28–37, 2004.

BUSINESS EXCHANGE. Video Game Industry. **Business Exchange**, 2011. Disponível em: <http://bx.businessweek.com/video-game-industry/>. Acesso em: junho de 2011.

COHN, M. **User Stories Applied: For Agile Software Development**. Boston: Addison-Wesley, 2004.

COHN, M. **Slides from Certified Scrum Master course**, 2008. [S.I.].

COCKBURN, A. **Agile Software Development**. 1ª ed. Reading, MA: Addison Wesley Longman, 2000.

CALELLE, D.; NEUFELD, E.; SCHNEIDER, K. Requirements engineering and the creative process in the video game industry. In: **13th IEEE International Conference on Requirements Engineering**. Proceedings... Washington, DC, USA: IEEE Computer Society, 2005. p. 240–252, 2005.

DEMACHY, T. 2003. Extreme game development: right on time, every time. **Gamasutra - The Art & Business of Making Games**, july 2003. Disponível em: <http://www.gamasutra.com/ resource_guide/20030714/demachy_01.shtml>. Acesso em: junho de 2011.

ESRB. Video Game Industry Statics. **Entertainment Software Rating Board**, 2011. Disponível em: <http://www.esrb.org/about/video-game-industry-statistics.jsp>. Acesso em: junho de 2011.

FLOOD, K. 2003. Game Unified Process (GUP). **GameDev.net**, may 2003. Disponível em: <http://www.gamedev.net/reference/articles/article1940.asp>. Acesso em: junho de 2011.

FOWLER, M. **Is Design Dead**? USA, july 2011. Disponível em: <http://www.ddj.com/dept/architect/184414721>. Acesso em: junho de 2011.

FULLERTON, T. A Playcentric Approach to Creating Innovative Games. **Game Design Workshop**, 2008.

GERSHENFELD, A.; LOPARCO, M.; BARAJAS, C. **Game Plan: the insider's guide to breaking in and succeeding in the computer and vieo game business**. New York: St. Martin' s Griffin Press, 2003.

GIBSON, A. **Agile game development and fun**. Technical report, University of Colorado Department of Computer Science, 2007.

HIGHSMITH, J. **Agile Software Development Ecosystems**. 1ª ed. USA: Addison Wesley, 2002.

KEITH, C. **Agile Game Development With Scrum**. 1ª ed. Addison-Wesley, 2010.

KNIBERG, H. **Scrum e XP Direto das Trincheiras**. 1ª ed. InfoQ Enterprise Software Development Community, C4Media, 2007.

KROUWEL, A. Extreme Game Development. **Develop - Game Design|Coding|Art |Sound|Business**, October 2010. Disponível em: <http://www.develop-online.net/features/1022/Extreme-Game-Development-pt1-killing-crunch>. Acesso em: junho de 2011.

MCGUIRE, R. Paper Burns: Game Design With Agile Methodologies. **Gamasutra - The Art & Business of Making Games**, june 2006. Disponível em: <http://www.gamasutra.com/features/20060628/mcguire_02.shtml>. Acesso em: junho de 2011.

MCSHAFFRY, M. **Game Coding Complete**. Scottsdale: Paraglyph Press, 2003.

PETRILLO, F. **Práticas Ágeis no Processo de Desenvolvimento de Jogos Eletrônicos**. 2008. Dissertação (Mestrado em Computação) - Universidade Federal do Rio Grande do Sul, . Orientador: Marcelo Soares Pimenta.

Petrillo, F., Pimenta, M., Trindade, F., Dietrich, C. What went wrong? A survey of problems in game development. **ACM Computer in Entertainment**, CIE: 7(1), February 2009.

PETRILLO, F. ;PIMENTA, M. S. Desenvolvimento de jogos é (quase) Ágil. In: Workshop Brasileiro de Métodos Ágeis (WBMA 2010), 2010, Porto Alegre. **CD-Anais do Workshop Brasileiro de Métodos Ágeis** (WBMA 2010), 2010.

POPPENDIECK, M.; POPPENDIECK, T. **Lean Software Development: An Agile Toolkit**. Addison Wesley, 2003. [S. l].

PRESSMAN, R. S. **Engenharia de Software**. 6ª ed. São Paulo: McGraw-Hill, 2006.

SCHOFIELD, B. Why Extreme Programming is Great for Game Development. **Gamasutra - The Art & Business of Making Games**, March 2007. Disponível em: <http://www.gamasutra.com/features/20070301/schofield_01.shtml>. Acesso em junho de 2011.

SCHWABER, K. **Agile Project Management with Scrum**. [S.l.]: Microsoft Press, 2004.

SCRUM ALLIANCE. **Scrum Alliance - transforming the world of work**, http://www.scrumalliance.org. Acesso em junho de 2011.

WIKIPEDIA. Video Game Industry. **Wikipedia, The Free Encyclopedia**, 2011. Disponível em: <http://en.wikipedia.org/wiki/Video_game_industry#History>. Acesso em: junho de 2011.

WIKIPEDIA. First-person shooter. **Wikipedia, The Free Encyclopedia**, 2011. Disponível em: < http://en.wikipedia.org/wiki/First-person_shooter>. Acesso em: junho de 2011.

APÊNDICE A <GAME DESIGN DOCUMENT DO JOGO SPYDER'S SAGA>

Este documento define em linhas gerais o jogo Spyder's Saga. Durante o restante do texto, o nome Spyder's Saga poderá ser será referenciado simplesmente por SS.

1. Definição do jogo

Spyder's Saga é um jogo para PC do tipo *platformer*. O jogador utiliza o teclado para controlar o personagem principal, a aranha. Nas fases, o jogador terá de eliminar inimigos, resolver pequenos *puzzles* e utilizar as teias da aranha para avançar até o final.

2. Estilo

Spyder's Saga é um jogo do tipo *platformer*, semelhante aos clássicos Mário World e Donkey Kong.

3. Core Gameplay

3.1 Câmera

Apesar do jogo ser implementado em um *engine* 3D, a visão que o jogador terá será em 2D. Ou seja, a coordenada Z da câmera será fixa. O jogador não irá controlar a câmera. A câmera irá acompanhar automaticamente o personagem principal.

A distância da câmera em relação ao plano onde a aranha irá interagir será típica dos jogos plataforma. O jogo terá dois planos: o primeiro, onde as ações do jogo irão se desenvolver, e o segundo, que será um plano de fundo, com árvores, pedras, etc.

3.2 Iteração do jogador

O jogador irá controlar a formiga através de fases ambientadas em florestas. Além da movimentação normal, a aranha poderá pular e jogar teias de aranha para eliminar inimigos e construir plataformas temporárias para chegar a locais de difiícil acesso. Adicionalmente, a movimentação da aranha será semi-realísitica, sendo que ela poderá andar na vertical e ficar de cabeça para baixo.

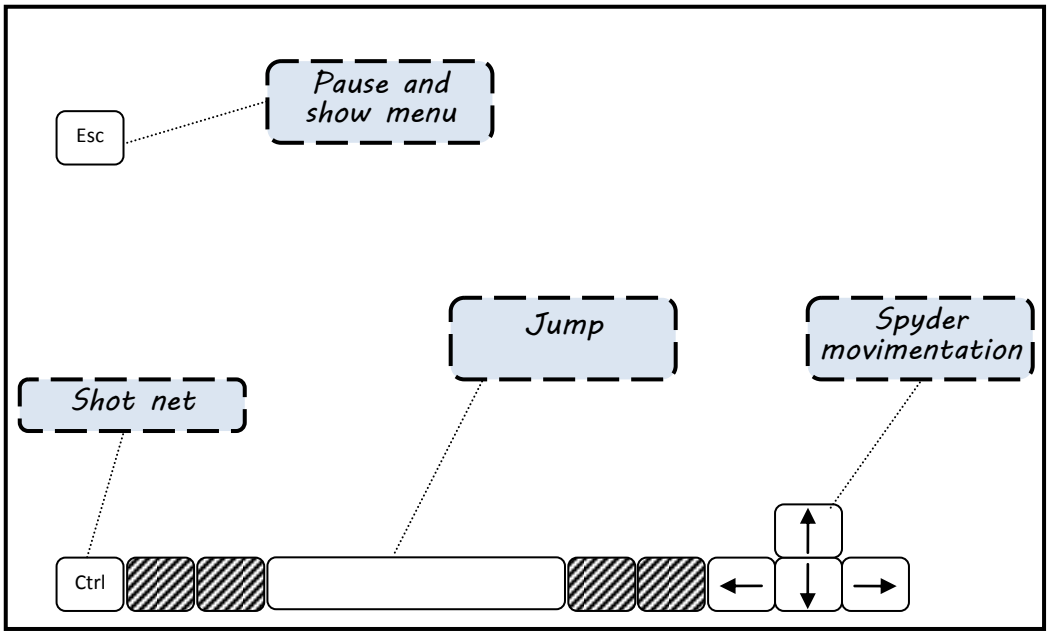
3.3 *Features* principais

As principais *features* que do jogo são:

- Movimentação semi-realística da aranha
- Ambientes divertidos em estilo *cartoon*
- Jogabilidade divertida e inovadora utilizando as teias da aranha

4. Controles

Os controles usados no jogo são todos pelo teclado, com exceção dos menus. A movimentação da aranha é pelas setas. A barra de espaço faz a aranha pular. O lançamento da teia é feito através da tecla Ctrl. Uma vez que a teia é lançada, apertar Ctrl novamente faz com que ela se abra, formando uma plataforma que desaparece em alguns segundos. Os controles poderão ser customizado no menu de opões. A figura A.1 mostra o layout padrão dos controles.



Fugura A.1 Layout dos controles

5. Público-alvo

O público-alvo é bem amplo, incluindo jogadores jovens, casuais e que gostam de jogos do tipo *platformer*.

6. Plataforma-alvo

O jogo será lançado para PC (*desktop*), sendo que está prevista uma versão de demonstração web.

7. Tecnologias

A principal tecnologia que será utilizada é o *Unity 3D*. Esse programa é classificado como um *Game Development Tool*, e inclui *engine 3D,* editor de código, gerenciador de projeto, etc. Todo o projeto será executado através do *Unity 3D*.

Para o desenvolvimento dos modelos 3D será usado o *software Blender*. As texturas serão desenvolvidas utilizando o *GIMP*. Nas versões originais do jogo, serão utilizados efeitos sonoros disponibilizados de forma *free* na internet. Nos estágios mais avançados de desenvolvimento, serão comprados pacotes de sons conforme a necessidade.

8. Distribuição

O jogo será distribuído através portais como o *Steam* e *Direct3D*.

APÊNDICE B <ARTEFATOS PRODUZIDOS NO DESENVOLVIMENTO DO ESTUDO DE CASO>

Neste apêndice, apresentamos alguns dos artefatos produzidos durante o desenvolvimento do estudo de caso. Acompanham comentários breves sobre a objetivo de cada artefato.

1. Diagramas de casos de uso

Com o objetivo de capturar os principais requisitos relacionados com o ator jogador e visualizá-los de forma clara, foi produzido um diagrama de casos de uso.

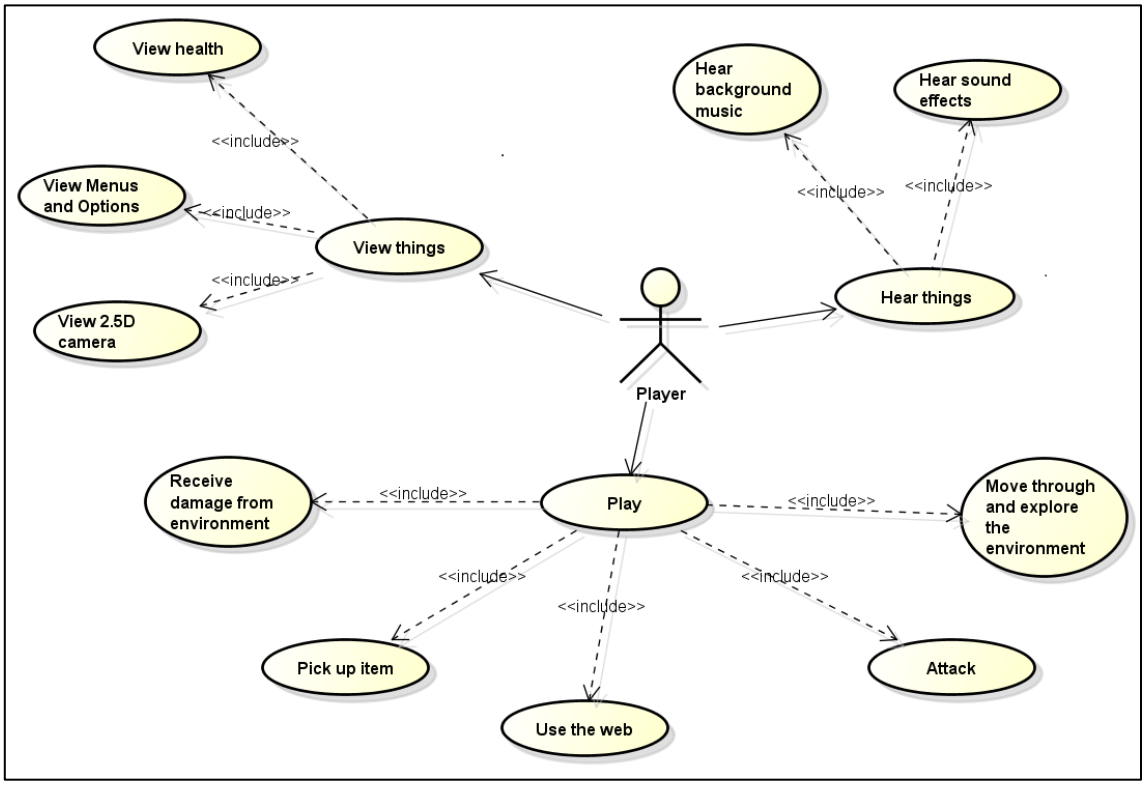


Figura B.1. Diagrama de casos de uso do ator jogador

2. Diagrama de *features*

Após a elaboração da documentação inicial, os principais requisitos foram representados através de um diagrama de *features*. Identificou-se uma hierarquia de *features*, ou *epics*, recurso valioso para visualizar a dependência entre as mesmas.

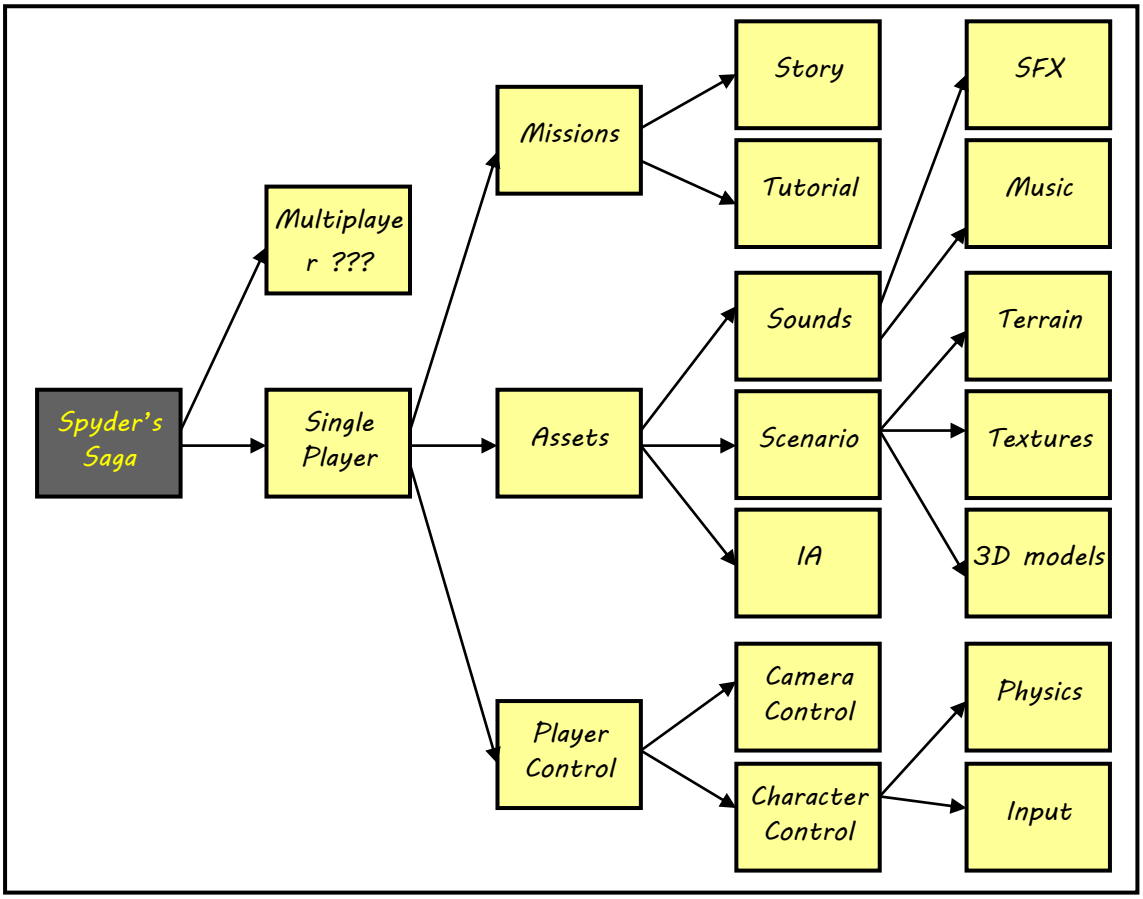


Figura B.2. Hierarquia de *epics*

3. Seleção das *features* que compõe o primeiro *release*

No início do planejamento do *release*, foram selecionadas as *features* que seriam desenvolvidas durante o mesmo. Esta seleção levou em conta o valor que cada *feature* representa para o jogador, bem como o objetivo do *release*, a saber, desenvolver uma versão jogável em que se pudesse testar o conceito do jogo.

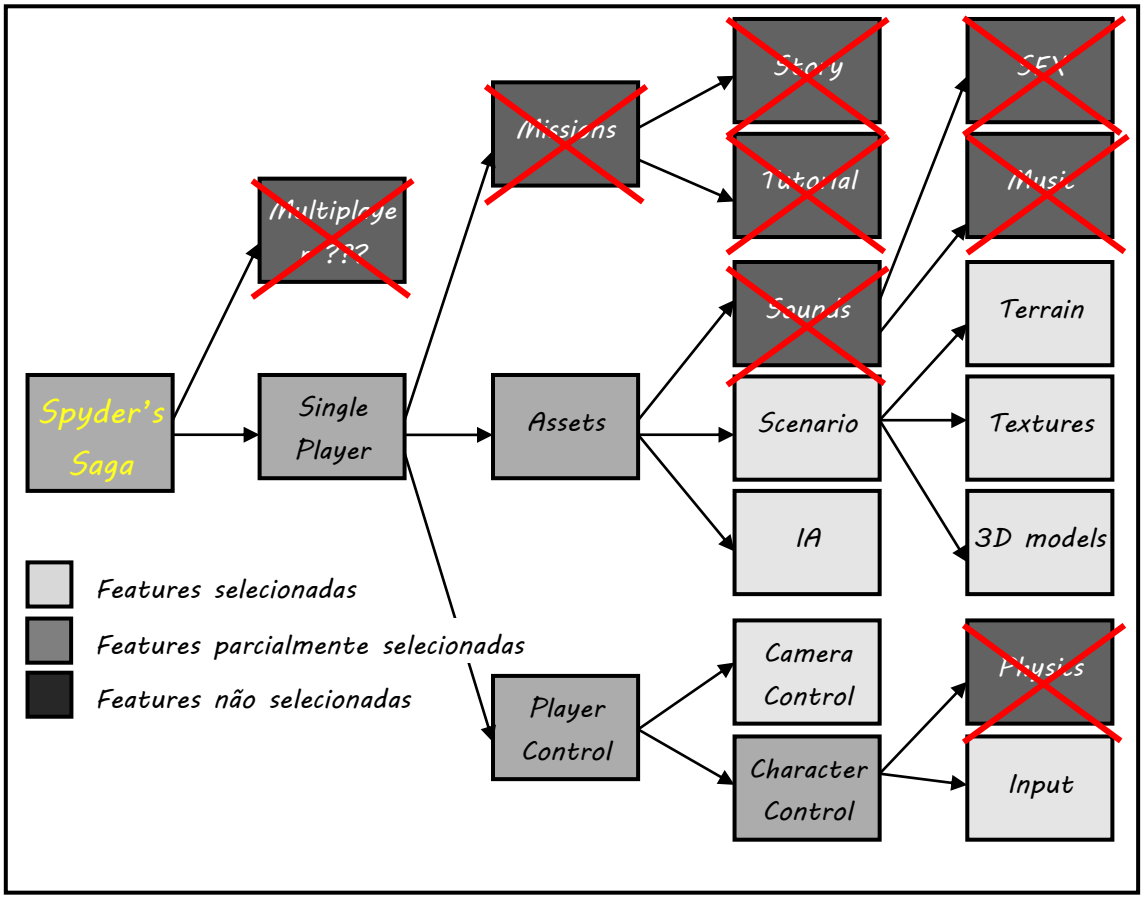


Figura B.3. Seleção das *epics* que compõe o primeiro *release*

4. Estórias de usuário

Após a definição de quais *features* seriam desenvolvidas durante o release, as mesmas foram detalhadas até ficar claro que precisava ser desenvolvido. Durante esse processo, as *epics* foram "quebradas" em estórias de usuário menores, na forma de *postits*.

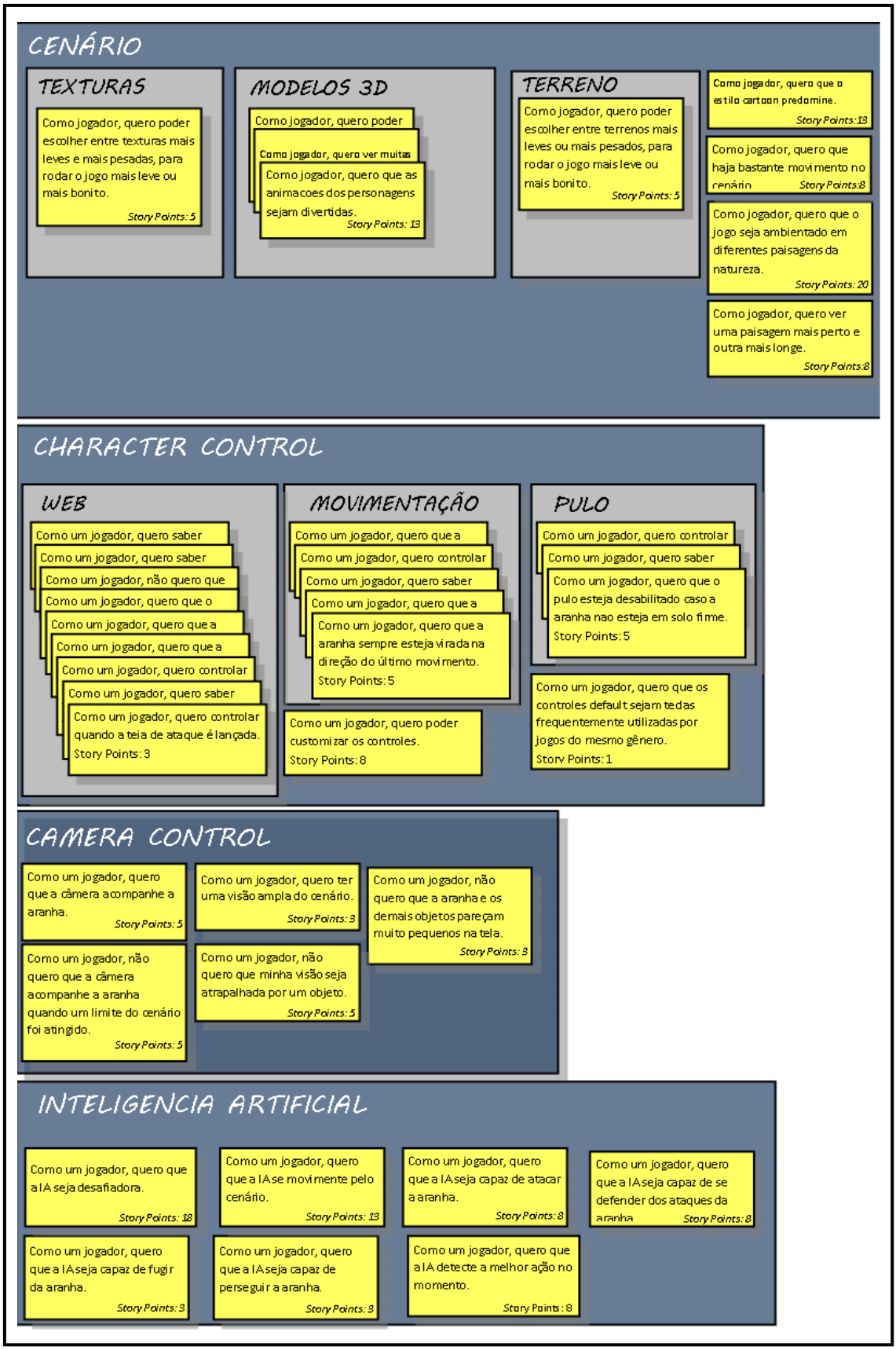


Figura B.4. Estórias de usuário

5. Distribuição das *features* entre os sprints

Após a elaboração das estórias de usuário das *features* selecionadas no *release* corrente, foi realizada a distribuição das mesmas entre os *sprints* que fazem parte do *release*. No estudo de caso, isso foi realizado utilizando diretamente as *epics*. Assim, o *sprint backlog* de cada *sprint* foi composto pelas estórias de usuário das *epics* respectivamente alocadas.

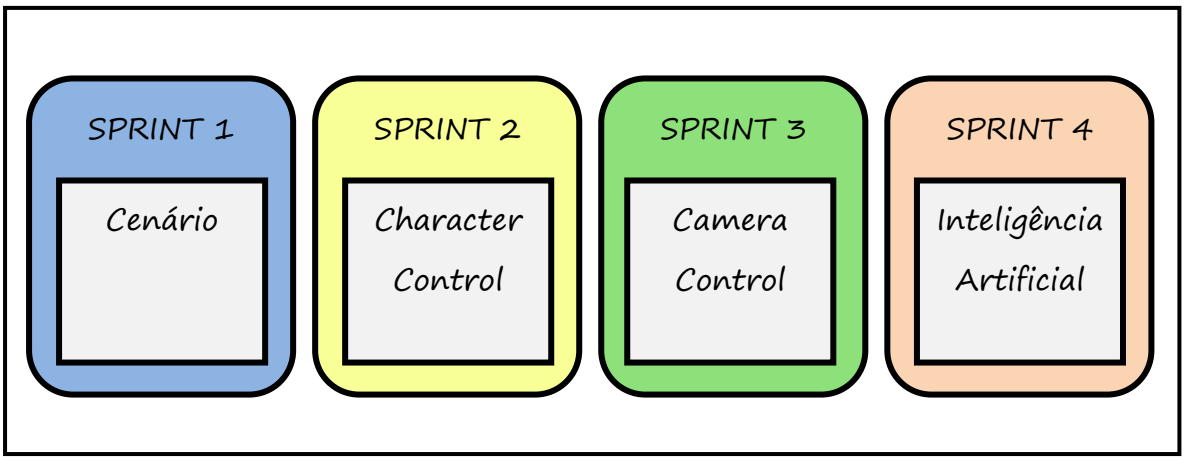


Figura B.5. Distribuição das *features* entre os sprints do primeiro *release*

6. Plano do *Release*

Finalizando a primeira visita a fase exploratória, foi elaborado o plano do *release*, que contém os prazos e objetivos dos *sprints*.

Plano do Release 1									
Release 1	Prioridade: 1	Sprint 1	Prioridade: 1	Sprint 2	Prioridade: 2	Sprint 3	Prioridade: 3	Sprint 4	Prioridade: 4
	Duração: 12 semanas		Duração: 3 semanas		Duração: 3 semanas		Duração: 1 semana		Duração: 3 semanas
	Início: 09/05 Fim: 26/06		Início: 09/05 Fim: 22/05		Início: 23/05 Fim: 05/06		Início: 06/06 Fim: 12/06		Início: 13/06 Fim: 26/06
	Keyword: primeira versão do jogo		Keyword: Cenário		Keyword: Controles do personagem		Keyword: Controles da camera		Keyword: IA
	Objetivo: desenvolver uma versão jogável do jogo que permita testar as principais features do mesmo.		Objetivo: desenvolver a primeira versão do cenário do jogo, composto por terreno, modelos e texturas. O cenário deve ser simplificado, porém condizente com a proposta do jogo (estilo cartoon).		Objetivo: desenvolver os aspectos principais referentes a interação do jogador com o personagem principal		Objetivo: desenvolver a primeira versão dos scripts que controlam a camera		Objetivo: desenvolver a versão inicial da IA, aplicando num inimigo com comportamento e ações simplificadas

Figura B.6. Plano do primeiro *release*