ESCOLA SUPERIOR DE TECNOLOGIA IPCA

Técnicas de Desenvolvimento de Jogos

Trabalho prático II

Trabalho realizado por:

- Carlos Alves 13219
- Bruno Rodrigues 13220

Ano letivo: 2017/2018

Docente: João Almeida

Curso: Engenharia Desenvolvimento de Jogos

Índice

Introdução

No âmbito da disciplina Técnicas de Desenvolvimento de Jogos, foi nos proposto realizar um projeto/jogo digital utilizando o ambiente de desenvolvimento integrado (IDE) *Visual Studio* e a ***FrameWork Monogame***.

Resumidamente, o ***MonoGame*** é uma implementação *open-source do Microsoft XNA* 4 Framework. Permite que desenvolvedores de jogos portem os seus jogos de *Windows*, *Windows Phone* e *Xbox* 360 para *iOS*, *Android*, *Mac OS X, Linux e Windows 8 Apps.*

Com este trabalho temos como objetivos melhorar as nossas habilidades na linguagem de programação – C# – também implementar grande parte do que foi apresentado em aula, e por último o nosso maior objetivo é desenvolver um jogo digital no ***Monogame*** *(FrameWork*), desde a construção de um GDD (*Game Design Document*), passando para a implementação dos algoritmos/código, de seguida o desenvolvimento da arte do jogo, isto é todas as *sprite's*, backgrounds e musica para o jogo e por fim talvez a sua publicação.

Neste relatório iremos descrever e explicar o desenrolar do desenvolvimento do jogo que realizamos durante o segundo semestre na disciplina de Técnicas Desenvolvimento de Jogos. Também iremos colocar os diversos problemas que foram nos aparecendo e de que maneira nós conseguimos resolver e ultrapassar estes mesmos problemas.

ESCOLA SUPERIOR DE TECNOLOGIA IPCA

Spin it! – 1º Jogo

Inicialmente, começamos por pensar como o nosso jogo/projeto seria elaborado. Foi necessário analisar alguns aspetos bastante importantes na criação do jogo como:

- Faixa etária - Qual grupo de pessoas o jogo seria direcionado;
- Plataformas – Qual a plataforma que irá ser possível jogar o jogo;
- Modelo de Monetização – Se seria pago ou gratuito;
- Tipo de jogo - Se iria ser um jogo de ação/aventura/puzzle/etc;
- Enredo – caso opta-se-mos por fazer um jogo com diálogos e história, era preciso desenvolve-la de acordo com os nossos objetivos e com os nossos conhecimentos.
- Personagens – Que tipo de personagem, se seria um objeto ou uma pessoa;
- As mecânicas do jogo;
- Os *Assets* necessários – Texturas 2D, *Sprite's*, Som e o código;
- Tempo necessário para implementar tudo.

Analisando bem, os aspetos acima referidos iniciamos o nosso *GDD (Game Design Document),* que consiste num documento "mestre" na criação de um jogo, ou seja é ele que contem todas as informações tanto no aspeto visual e conceitual do jogo, como também tecnológico. Este documento é deveras muito importante na criação de um jogo, pois é nele que nós nos vamos guiar à medida que vamos fazendo o jogo.

Com o dissecar dos aspetos referidos anteriormente decidimos que o nosso jogo iria consistir num jogo dirigido para qualquer tipo de idades, talvez mais focado em jovens/adultos. O jogo estaria disponível gratuitamente para computadores (*Windows*) e para *smarphones (Android/WindowsPhone*). Também achamos que um aspeto muito importante que queríamos no nosso jogo é que tudo o que fosse utilizado no projeto/jogo – código, texturas, *sprites* e som – seriam feitos apenas por nós. Sendo que o jogo não teria qualquer problema com *copyright* e achamos que desta forma iriamos ter um experiencia "completa" na criação do jogo.

O projeto "*Spin it!*" consiste num jogo onde o utilizador se encontra a controlar um taco/bastão, que se mantem fixo num ponto do ecrã e similarmente roda conforme o utilizador desejar. Tem como objetivo rebater todas as bolas que venham em sua direção, pois as bolas irão aparecer aleatoriamente na tela e irão ao encontro do taco de forma que o jogador possa rebate-las com o taco. Conforme for rebatendo as bolas, vai acumulando pontos – que poderão ser utilizados mais tarde numa "loja" dentro do jogo, onde se encontrarão diversos tacos para serem

comprados. Caso uma das bolas acabe por colidir na parte de baixo do taco, onde estariam as mãos do jogador, este verá uma mensagem a informar que perdera o jogo e será exibido um placar com a pontuação que o jogador obteve naquela partida.

Após feito o *GDD* (*Game Design Document*), começamos por estabelecer prazos e dividir tarefas, pois sendo dois elementos não queríamos que ambos tivessem a trabalhar na mesma coisa, desta forma iriamos economizar tempo.

Implementação

Primeiramente, decidimos começar por implementar a mecânica principal que consiste em colocar o taco no meio do ecrã de jogo (fixa-lo), e fazer com que o taco rodasse a partir da pega mas que roda-se num só ponto do ecrã.

Para implementar a mecânica descrita anteriormente, criamos as variáveis necessárias para carregar o taco, roda-lo e dar lhe uma posição – variável *position* do tipo Vector2, variável *Player* do tipo Texture2D e outra *angle* do tipo *float*. De seguida, na função ***LoadContent***() – que tem como utilidade carregar todo o conteúdo ou recursos do projeto, como a arte do jogo e o áudio – inserimos a linha de código exibida na imagem a baixo. Nela fazemos uso da variável *Player* que criamos anteriormente e carregamos a imagem do taco que se encontra na pasta *Content*.

```
// carregar conteudo da pasta content
Player = Content.Load<Texture2D>("taco.png");
```

Fig.1

Com isto o taco já aparecia no ecrã, agora precisávamos apenas de coloca-lo a rodar a partir da pega do taco. Para isso, na função ***Update***() - responsável por procurar os dispositivos de entrada – modificamos uma parte do código para que fosse possível rodar o taco com o rato. Estas alterações podem ser vistas na figura nº2 e na figura nº3.

```
// TODO: Add your update logic here
MouseState curMouse = Mouse.GetState();
Vector2 mouseLoc = new Vector2(curMouse.X, curMouse.Y);

Vector2 direction = mouseLoc - Position;

angle = (float)(Math.Atan2(direction.Y, direction.X));
```

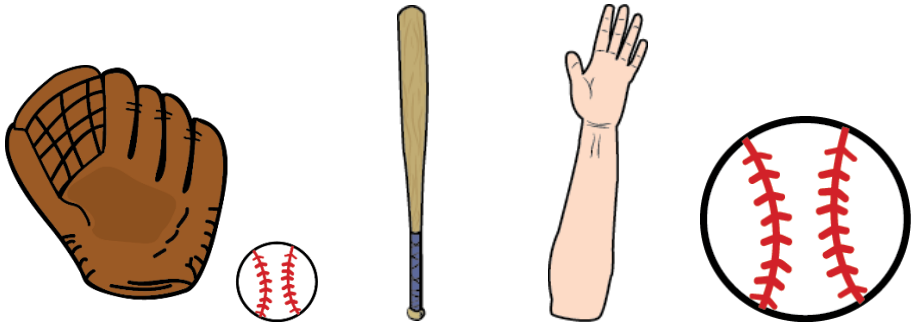
Fig.2

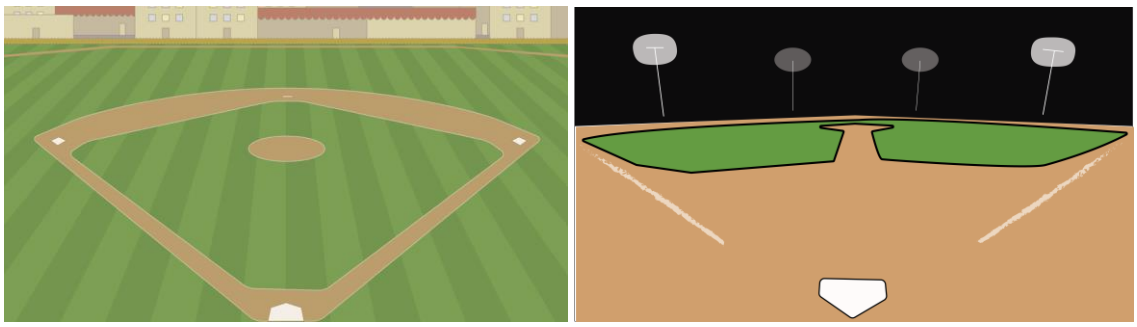
```
spriteBatch.Draw(Player, new Rectangle((int)Position.X,
    (int)Position.Y, Player.Width, Player.Height),
    null, Color.White, angle,
    new Vector2(Player.Width / 2, Player.Height / 6),
    SpriteEffects.None, 0);
```

Fig.3

Completada e implementada a primeira mecânica do jogo, começamos a desenvolver a parte gráfica do jogo, desenhar as texturas 2D necessárias para inserir no jogo. Para desenvolver a arte gráfica do jogo recorremos ao *software Illustrator CC 2015* usado numa das disciplinas do primeiro semestre do curso de jogos. Na arte optamos por fazer algo que se aproxima-se do ***cartoon***, um estilo arrojado com contornos negros e grossos e com um preenchimento solido.

Abaixo seguem alguns exemplos de texturas 2D e backgrounds que foram feitas por nós para este jogo/projeto.





Visto que ainda nos encontrávamos dentro do cronograma e com a implementação da primeira mecânica e com a arte gráfica pronta, avançamos de imediato para a implementação da segunda mecânica que nos pareceu ser a mais difícil de colocar no projeto, esta mecânica consistia em inserir as bolas e as suas colisões no jogo. Primeiramente para carregar as bolas, fizemos na mesma forma sequencial de como fizemos com a inserção do taco, expeto que as bolas não iriam ser carregas apenas num só ponto e sim de vários pontos aleatórios, de preferência nas extremidades do ecrã do jogo. E não iam rodar, apenas se moviam em linha reta até ao taco (centro do ecrã). Para que elas fizessem esse movimento até ao taco, usamos uma serie de comandos que se encontravam no livro *Learn 2D Game Development with C-sharp* , com as funções que encontramos num dos capítulos deste livro, adaptamos as para que pudéssemos inserir as bolas da maneira que nós desejamos.

Após a inserção das bolas aleatórias e a trajetória que iriam percorrer implementadas era preciso implementar as colisões no taco ou nas bolas. Decidimos fazer no taco, pois assim era apenas um objeto que iria sofrer alterações. Inicialmente, tentamos fazer as colisões por retângulos – que consiste em criar retângulos sobre as texturas 2D que neste caso seria o taco e as bolas – mas visto que os nossos objetos não tinham qualquer formato retangular, este tipo de colisões à partida não iriam resultar da forma que queríamos mas como era simples e fácil de implementar fizemos mesmo assim e experimentamos como reagiam as bolas ao bater no taco.

Podemos observar que o taco estando parado (sem rodar), as bolas colidiam com o taco mas de uma maneira estranha. Exemplo a bola ainda não tinha chocado com o taco e já estava a ser redirecionada para outro lado da tela. Mais tarde deparamo-nos que o tamanho do quadrado era um pouco maior ao tamanho da *sprite*, desta forma resolvemos este problemas mas ao testar continuadamente as colisões reparamos que se o taco tivesse a rodar enquanto as bolas passavam por ele as colisões desapareciam e as bolas não faziam ricochete no taco como deveria fazer. Com bastante pesquisa, encontramos uma informação bastante útil sobre colisões no ***Monogame***, alertava que caso o objeto se movesse demasiado rápido ele não iria colidir com o outro objeto. Visto isto, decidimos utilizar outro tipo de colisão, a colisão por *pixels*, consiste basicamente numa colisão de ambiente 2D precisa e é feita através de uma cor.

Para utilizar este tipo de colisão foi necessário algumas alterações nas *sprites*, pois necessitava que os contornos fossem mais grossos de maneira que a colisão se fizesse sem quais problemas. Infelizmente ao implementar este tipo de colisões tivemos ainda mais problemas do que em relação à colisão anterior. As bolas ao embaterem no taco nem sempre se observava a colisão, pois a bola passava direto pelo taco, outras bolas embatiam no taco mas o taco tinha de esta imóvel e mesmo estando imóvel grande parte das bolas ao embater no taco não faziam ricochete mas sim deslisavam pelo taco.

A partir deste momento fomos incapazes de progredir no projeto **Spin It!** e visto que o tempo era bastante reduzido decidimos começar um projeto do “zero”.

Devido aos problemas descritos acima, começamos um projeto de novo, sendo que foi necessário realizar todos os "passos" realizados no projeto anterior, desde a construção da *GDD* (*Game Design Document*), da máquina de estados do jogo, mecânicas entre outras tarefas.

Inicialmente, começamos por formalizar uma ideia para o jogo, analisar os recursos que já possuíamos e o tempo restante, que já era reduzido. Depois de bem analisado, decidimos que o jogo seria do tipo *sidescroller* (onde a camera é somente lateral, ou seja, só vemos um lado do ambiente). Escolhemos este tipo de jogo, pois seria mais simples de implementar e era o melhor tipo de jogo que podíamos escolher devido ao tempo restante para entregar o projeto.

Tendo as ideias formalizadas, iniciamos uma pequena pesquisa para reunir alguns recursos e algumas informações sobre *sidescrollers*.

De seguida, implementamos as classes fundamentais a um *sidescroller*, uma que trata de colocar a imagem de fundo em *loop*, a classe **ParallaxingBackground**, e claro, a classe **player**, para pôr o jogador.

Quando decidimos que tema que o nosso jogo iria retratar, um jogo com ambiente no espaço, em que o objetivo era ter a máxima pontuação possível, derrotando os inimigos e eliminando os bosses.

Criamos então a classe **Enemy**, e como queríamos que tanto as *sprites* do jogador como as *sprites* do inimigo fossem animadas, criamos a classe **Animation**.

De seguida, adicionamos os projeteis, para o jogador e para os inimigos, e adicionamos também a classe **Boss**.

Foram criadas colisões, tanto entre projeteis e jogador, projeteis e inimigos, projeteis e asteroides (descrito posteriormente), jogador e inimigos, jogador e boss, projeteis e boss e projeteis e asteroides.

Seguidamente, como os inimigos eram pouco variados, e relativamente fáceis de derrotar, implementamos obstáculos, que neste caso, são asteroides, que mais tarde, foram modificados para largar *Packs* de vida, quando a classe **Health Pack** foi adicionada.

Mais para o fim, adicionamos os menus do jogo, com um Menu Principal, pausa, *GameOver*.

Relativamente à questão da arte, foi utilizado o *software* ***Illustrator CC 2015*** na construção das *sprites*, dos botões e até mesmo dos backgrounds. Com este *software* foi possível que até mesmo nós alunos com pouco experiencia a desenhar, conseguimos construir bons cenários, boas *sprites*, e algumas texturas 2D que nos ajudaram bastante na construção do jogo.

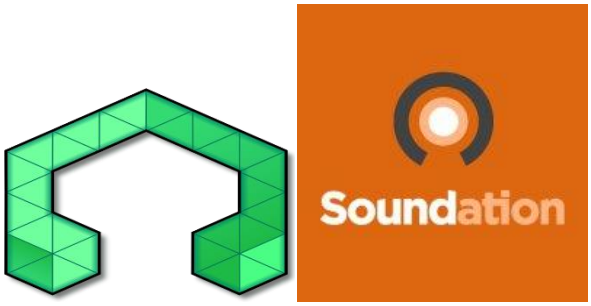
Na construção das *sprites* tivemos diversos problemas, nos quais não conseguimos resolver devido a falta de conhecimento. Nas *sprites* ao colocar no jogo, as naves comportavam-se de forma irregular, pois na construção das mesmas, elas não estavam completamente centradas e alinhas causando nos vários problemas durante a realização do jogo.

Para que as colisões funcionassem "perfeitamente" aumentamos os contornos de todas as texturas 2D, de forma que o *pixel* fosse reconhecido com alguma facilidade.

Abaixo seguem algumas das *Sprites*/ botões realizados para este projeto:

Som

Com o auxílio dos *softwares* **LMMS** e do **Soundation Studio**, foi possível realizarmos pequenos trechos musicais que serviram como musica ambiente no jogo. Foram feitas duas músicas, no projeto completo, mas apenas uma foi utilizada como música principal do jogo.



Código

Class Animation – Responsável pela divisão das *sprites* das diversas texturas 2D, esta classe trata de toda animação do jogo, isto funciona da seguinte maneira: o programador insere algumas informações sobre a *sprite* em questão, por exemplo, a altura e comprimento o nº de *frames* que a *sprite* possui. Com estas informações inseridas a classe vai transitar por cada *frame* a uma determinada velocidade, esta velocidade também inserida pelo programador.

Classe Asteroid – Esta classe é responsável pelos asteroides (obstáculos) que aparecem durante a partida. Nesta classe colocamos o asteroide como se fosse um organismo "vivo", pois demos-lhe vida, esta vida que tem um x valor, caso ela fique inferior a zero durante a partida, isto significa que o asteroide "morreu" fazendo com que este desapareça. Também tem como função dar uma velocidade ao asteroide e um dano físico quando embate contra algo.

Classe Boss – Classe responsável pelo inimigo "*boss*", o inimigo final de cada estágio. Este por sua vez funciona como o asteroide, pois possui diversas variáveis, como a vida, o dano que inflige, os pontos de score, a posição em que aparecerá no jogo e quando ele tem de desaparecer.

Classe Enemy – Classe responsável pelos inimigos (*minions*), tem igualmente as mesmas, ou parecidas variáveis que a classe **boss** e **asteroid**, pois trata-se de *NPC's*.

Classe Enemy Projetil – Classe que de uma forma ou outra está diretamente relacionada com a classe **Enemy**, pois nesta classe são trabalhados os projeteis (tiros) que os inimigos lançam durante a partida, sendo que eles, a causar mais dano no jogador principal.

Classe GUIElements – Classe que auxilia na criação dos menus presentes no jogo, desenha os botões.

Classe HealthPack- Classe responsável por inserir uma texture 2D, capaz de melhorar e aumentar a vida do jogador principal.

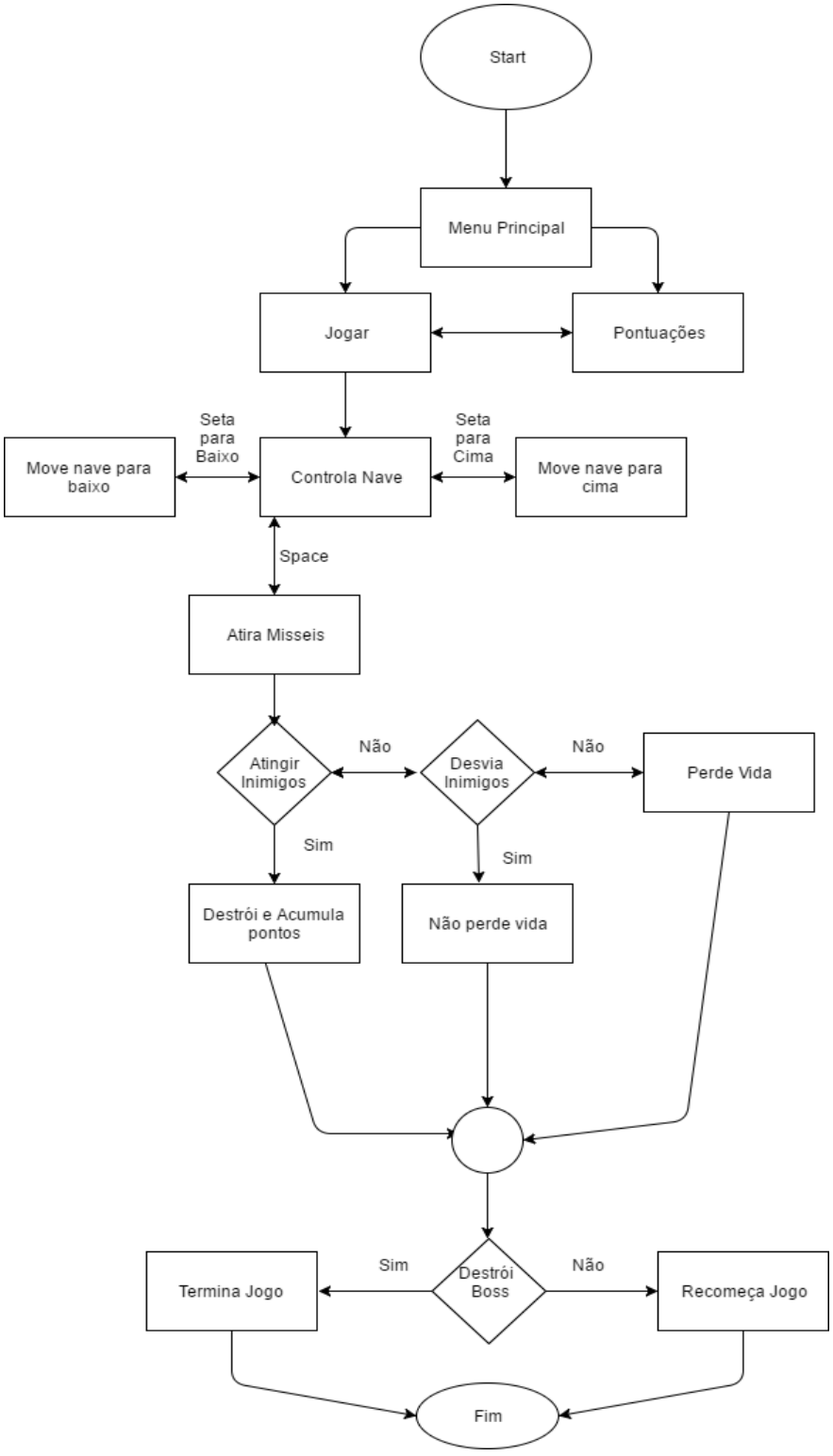
Classe Invencível – Classe semelhante à anterior, apenas difere no facto de ser uma textura 2D que cria um escudo para proteger o jogador.

Classe ParallaxingBackgroud – Responsável por armazenar um serie de imagens num *array* e fazer com elas se movimentem de forma linear, transparecendo a sensação ao utilizador que esta se está a mover.

Classe Player – Classe responsável por "caracterizar/formar" a Texture2D do jogador principal, que por sinal é uma nave espacial. Aqui encontram-se todas as informações cruciais da textura 2D como: a vida, animação e posição que deverá aparecer na partida.

Game1 – Classe mais importante de todas e a classe "principal", onde é feito o uso de todas as classes descritas anteriormente. São desenhados os menus como também as colisões, também como os limites do mapa, entre outras coisas, como a inicialização de todas as classes anteriores.

Máquina de estados e Diagrama de Blocos



Problemas

Um dos maiores problemas ao escrever o código foi a implementação dos menus, e o que estes implicavam, também a utilização correta da classe que animava o jogo foi deveras problemático.

Na implementação de cada menu, cada condição tinha de ser estudada, pois caso o jogador tivesse perdido, tinha que se chamar a condição *GameOver*, e reiniciar todas as variáveis a zero, limpar todas as listas, isto só se o jogador perdesse.

Caso fosse do menu pausa para o menu principal, tinha que ser tudo reiniciado, desativar a pausa, o que no momento do desenvolvimento gerou alguma confusão.

Outro problema foi ao inserir as *sprites* no jogo, pois muitas vezes a *sprite*, quando colocada no jogo aparecia aos "soluços", isto é, as imagens não se encontravam centradas o suficiente para passarem com fluidez, o que era complicado de acertar, visto que a *sprite* tinha de estar desenhada exatamente no centro de cada janela da *spritesheet*. Este problema foi resolvido à base de tentativa e erro, mas este processo foi muito moroso, o que nos prejudicou posteriormente.

Outros detalhes menores também foram complicadas de resolver, mas que rapidamente se chegou a uma resolução.

Ao desenvolver o código, algumas coisas não foram possíveis de implementar devido a falta de tempo, como também devido a falta de conhecimentos.

Melhorias

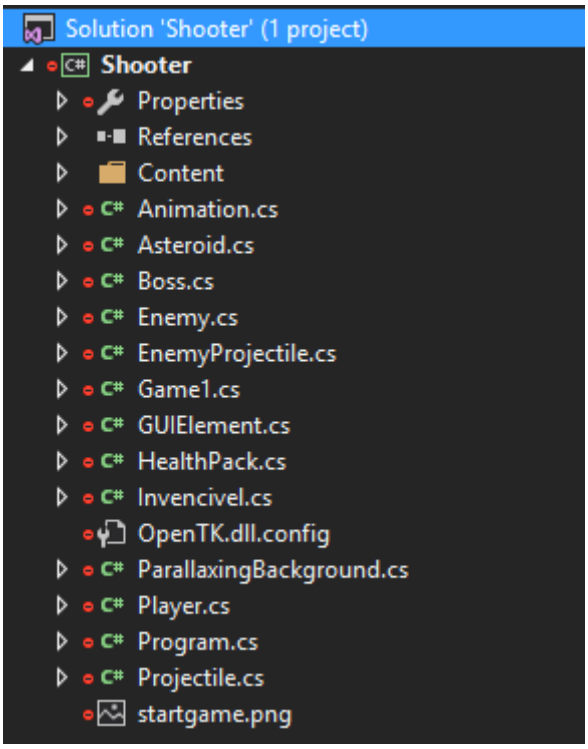
Devido ao tempo ter-se tornado "curto" por não conseguirmos realizar o primeiro jogo e termos começado outro, não conseguimos implementar diversas melhorias no código e do jogo.

Tais melhorias ao código passariam por melhorar as colisões entre elementos de jogo, uma maior variedade de inimigos, que seriam desbloqueados à medida que os elementos do jogo mais difíceis fossem derrotados (boss).

Uma das ideias mais ambiciosas passaria por adicionar uma loja ao jogo, onde seriam disponibilizados ao jogador mais naves, diferentes tipos de projeteis, explosões, músicas de fundo, entre outros aspetos visuais.

Por último, a ideia de criar um menu de opções, onde o jogador poderia diminuir o nível do som da música ambiente ou até mesmo desligar completamente o som.

Estes são alguns dos elementos que se poderão vir a implementar no futuro.



Conclusão

Concluímos que com a realização deste trabalho prático na disciplina de Técnicas de Desenvolvimento de Jogos, foi possível, com imensa pesquisa e trabalho, criar um jogo desde o zero. Começando pela ideia, passada para o GDD, depois a implementação do código, a criação de arte (*sprites*, texturas 2D e sons) e terminando com a realização de um trailer. Futuramente poderemos vir a inserir o jogo no mercado (gratuitamente).

Com este trabalho foi possível melhorarmos as nossas habilidades, tanto na linguagem de programação C#, na criação de arte e também melhorar as nossas capacidades para ultrapassar os problemas e adversidades.

Devido a diversos problemas encontrados durante a realização do jogo, e também à quantidade de trabalhos neste semestre, não conseguimos implementar certas mecânicas que gostaríamos de implementar. Estas encontram-se descritas na secção “melhorias”.

Por fim, concluímos que através da FrameWork MonoGame e do VisualStudio, ambos são excelentes ferramentas que tornam possível criar imensos projetos de diversas maneiras.

Bibliografia

Jebediah Pavleas, J. K.-W. (2013). *Learn 2D Game Development with C#* (Vol. 1). U.S.A: Apress.

Micorsoft. (s.d.). *msdn Microsoft*. Obtido de Microsoft : https://msdn.microsoft.com/en-us/library/microsoft.xna.framework.game.initialize.aspx

Team, M. (2 de Setembro de 2009). *MonoGame* . Obtido de MonoGame : http://www.monogame.net/