POLITÉCNICO DE LEIRIA | ESCOLA SUPERIOR DE TECNOLOGIA E GESTÃO

Bachelor Degree in Games and Multimedia

Advanced Game Programming Topics

Project 1

2D Game Engine with Xenon clone

Autores:

Francisco Domingues - 2162370

Miguel Monteiro - 2171005

Professor:

Gustavo Reis

Introdução

Para este projeto, foi-nos pedido que criássemos uma game engine de raiz em C++ utilizando os aplicativos *SDL2* e *Box2D* de modo a podermos desenvolver um jogo nesta mesma engine.

Neste relatório, iremos detalhar todos os algoritmos que desenvolvemos para tornar isto possível, as nossas justificações por detrás do código que implementamos, os objetivos que atingimos neste projeto, aqueles que não atingimos e a lista de todos os sites e referências que pesquisamos para este projeto.

Game Engine

Objetivos:

Alcançados:

- Gamepad controller support
- Event Handling
- Game Loop
- Resource Management
- Level
- Actor
- Pawn
- SDL2 Wrapper classes
- Box2D classes
- Animation

Xenon Clone (jogo)

Objetivos:

Alcançados:

- Resource Management
- Spaceship
- Missiles
- Enemies
- Level

Parcialmente alcançados:

- Spaceship
- Missiles

Não alcançados:

- Enemy Projectiles
- Vertical Scroller

Implemented Algorithms

Engine

Gamepad Controller Support

Para que se possa usar um *gamepad controller* para controlar um objeto no jogo (neste caso a nave), foi implementada a classe “GameController” que usa funções do *SDL2* para receber o *input* do controller que depois pode ser acedido pelo “Pawn”. Neste caso usámos o *input* do analógico esquerdo para o movimento do personagem.

Game Loop

O Game Loop encontra-se dentro da classe da Engine e é o que executa a lógica do jogo. Começa por calcular o *deltaTime* (tempo desde a última execução) e, de seguida, atualiza a física no *level* e lida com os *events* do *SDL*. Depois disto, corre o *update* de cada *actor* através do *level* e finalmente desenha os mesmos no ecrã. Por fim calcula o *frametime*, executa animações e espera pelo próximo *frame*.

Resource Management

Nas várias classes do projeto tentámos ter cuidado com a gestão da memória, tentando-a libertar assim que já não seja necessária. Um exemplo notável é no caso do *level* e dos *actors*: o level vai guardando os *actors* que são instanciados e, ao terminar o nivel, destroi-os antes de se destruir a sí.

Level

Para tarefas como armazenar listas de *actors* e *animations* e a criação do mundo do Box2D (onde são criados os corpos dos objetos e simuladas as físicas) foi criada a classe “Level”.

Esta classe é usada como variável na classe “Engine” de forma a que as suas funções possam ser chamadas no game loop, na altura adequada.

Actor

Para implementar uma classe que servisse de base para os objetos criados no jogo, foi criada a classe “Actor” que guarda uma posição em *x* e em *y* e ainda variáveis para a *texture*, *tilemap* e *animation*. Nesta classe foram criadas as funções “update()” e “render()” que são chamadas durante o game loop para atualizar tanto a posição do objeto, como a representação gráfica do mesmo.

Pawn

Foi criada uma class pawn para servir de base para o “player”. Esta class controla o movimento do personagem jogável e é flexível de modo a poder ser estendida e reimplementada.

SDL2 Wrapper classes

Para manusear e utilizar métodos do *SDL2*, foram criadas várias classes para generalizar vários termos como *“Window”*, *“Texture”*, etc. Estes permitem ao game loop e ao jogo não ter contacto direto com o *SDL*, reduzindo a complexidade e aumentando a portabilidade.

Box2D Classes

De forma a poderem ser chamadas funções ao ser detectada uma colisão, foi criada a classe “ContactListener” que estende da classe “b2ContactListener” do Box2D que reimplementa a função “BeginContact” que é chamada no momento em que se inicia um contacto entre dois corpos do Box2D. Para ser usada nesta função foi criada a classe “ContactSensor” que tem a função virtual “onContact” que é chamada aqui sempre que um “ContactSensor” estiver presente num contacto.

A classe “ContactSensor” serve, por sua vez, para servir de *parent class* para as classes em que queremos que haja algum código a ser executado quando uma colisão é detectada.

Para criar o corpo no qual as físicas vão ser simuladas foi criada a classe “RigidBody” que é usada nas classes em que queremos criar um corpo usando o Box2D.

Por último, foi criada também a classe “Projectile” que serve para criar um corpo que tenha as propriedades de um *bullet* do Box2D. Nesta classe é criado um “RigidBody” e é definida a velocidade a que esse corpo se começa a mover quando é criado.

Animation

A implementação de animações foi criada através das classes “Texture”, “Tilemap” e “Animation”. Na classe “Texture” são usadas funções do sdl2 para pegar num ficheiro de formato.bmp e criar uma “SDL_Texture” e são ainda guardados dois “SDL_Rect” para serem usados pelo renderer para desenhar a textura no ecrã.

Na classe “Tilemap”, através de uma referência à textura, e ao ser especificado o número de linhas e colunas do *tilemap*, é definido o tamanho de cada *tile*. Finalmente, no construtor da classe “Animation” é especificado um “Tilemap”, a ordem de animação das *tiles* pretendida e um valor *boolean* correspondente a se queremos uma animação em *loop* ou não.

Game

Spaceship

A “Spaceship” é o nosso jogador. Ela estende da classe “Pawn” e é composta pelas texturas e animações necessárias além de receber o input do jogador para se mover.

Missiles

No caso da classe *“Missile”*, foram atribuídas as classes *“Actor”* e *“Projectile”* como *parent classes*, de forma a que o míssil possa ter uma textura, possa implementar a função *“onContact()”* e seja criado um *rigidbody* com as propriedades de um míssil, com valores como a posição, tamanho, densidade e velocidade a serem passados pelo construtor da classe.

Enemies

Para implementar inimigos no jogo foi criada a classe *“Enemy”* (*child class* de Actor) que serve de *parent class* para as classes *“Loner”* e *“Rusher”* e que guarda variáveis que são usadas para criar o *“Tilemap”*, a *“Animation”* e o *“RigidBody”*. Nestas classes é depois, no construtor, atribuída a respectiva textura, *tilemap* e é definida a animação

correspondente ao inimigo e criado um corpo dinâmico do *Box2D* ao qual é atribuída uma velocidade em *x* no *"Loner"* e em *y* no *"Rusher"*.

Enemy Projectiles

Os projéteis dos inimigos não foram implementados neste projeto, mas caso fossem, a abordagem seria semelhante à usada para a classe "Missile", usando as *parent classes* "Actor" e "Projectile".

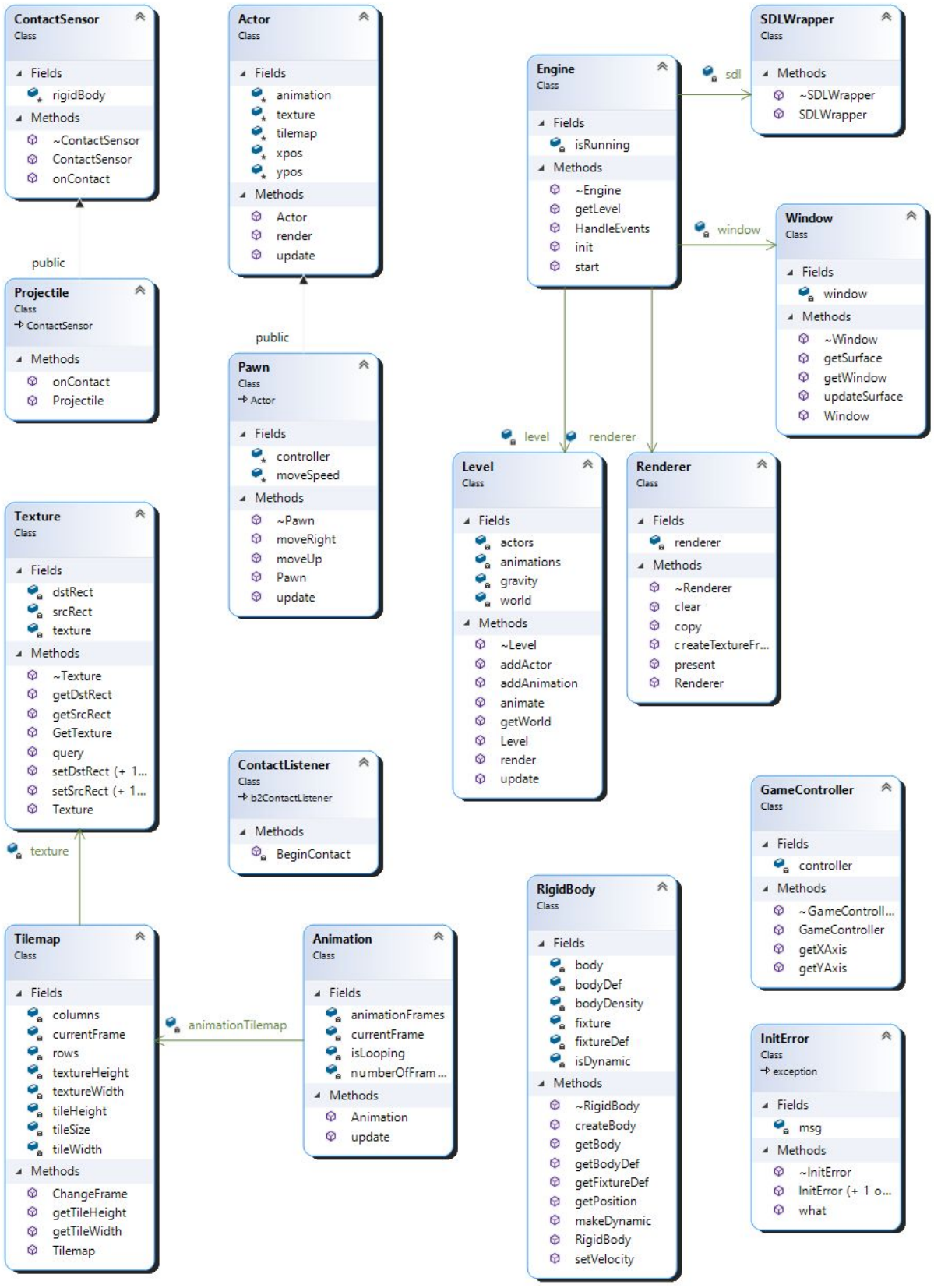
Vertical Scroller

O Vertical Scroller não foi implementado.

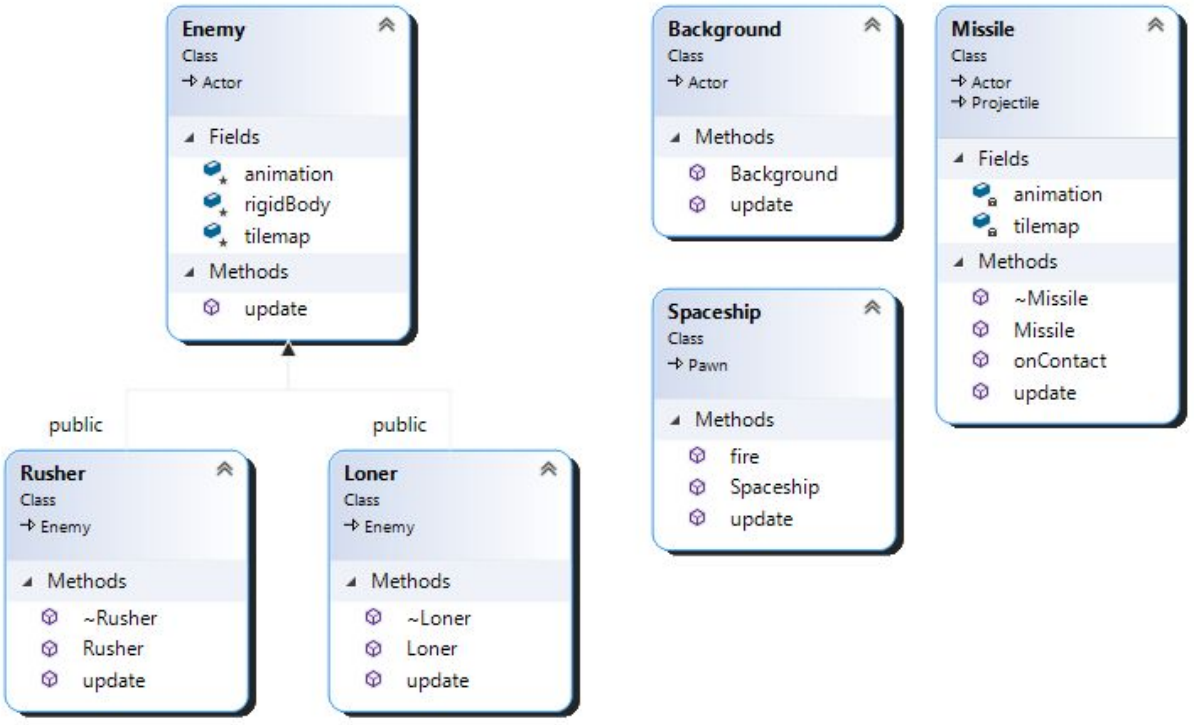
Level

O ficheiro "Xenon.cpp" foi usado para criar o nível do jogo, onde é criada uma variável engine, que depois é usada para chamar a função "init()" com o nome e dimensão da janela. De seguida é criado o *background*, os inimigos e a nave, que depois vão ser geridos no game loop que se encontra na função "start()" do engine que é chamada a seguir.

Engine Class Diagram



Xenon Class Diagram



Conclusão

O resultado obtido e objetivos alcançados não atingiram todas as nossas expectativas, mas foram uma boa ferramenta de aprendizagem.

Não ficamos totalmente satisfeitos com o resultado final, devido a alguns imprevistos e dificuldades que causaram atrasos no desenvolvimento do projeto.

Acreditamos ter uma base sólida para continuar e completar a game engine com o segundo projeto da UC.

Referências

- “Breakout.” *LearnOpenGL*, learnopengl.com/In-Practice/2D-Game/Breakout.
- Catto, Erin. *Box2D Documentation*, https://box2d.Org/Documentation/.
- Let's Make Games. “How To Make A Game In C++ & SDL2 From Scratch!” *YouTube*, www.youtube.com/playlist?list=PLhfAbcv9cehhkG7ZQK0nfIGJC_C-wSLrx.
- *SDL Wiki*, wiki.libsdl.org/.
- Fichas resolvidas nas aulas da unidade curricular.