Technical Design Document

Team MKEA

Contents

Target platform:

We decided to build our game for Windows PC as all of us are familiar with a PC and it would make development and testing easier. Specifically, Windows 10.

Development Environment and Tools:

- Visual Studio Enterprise 2017
- C++ Standard Template Library (STL)

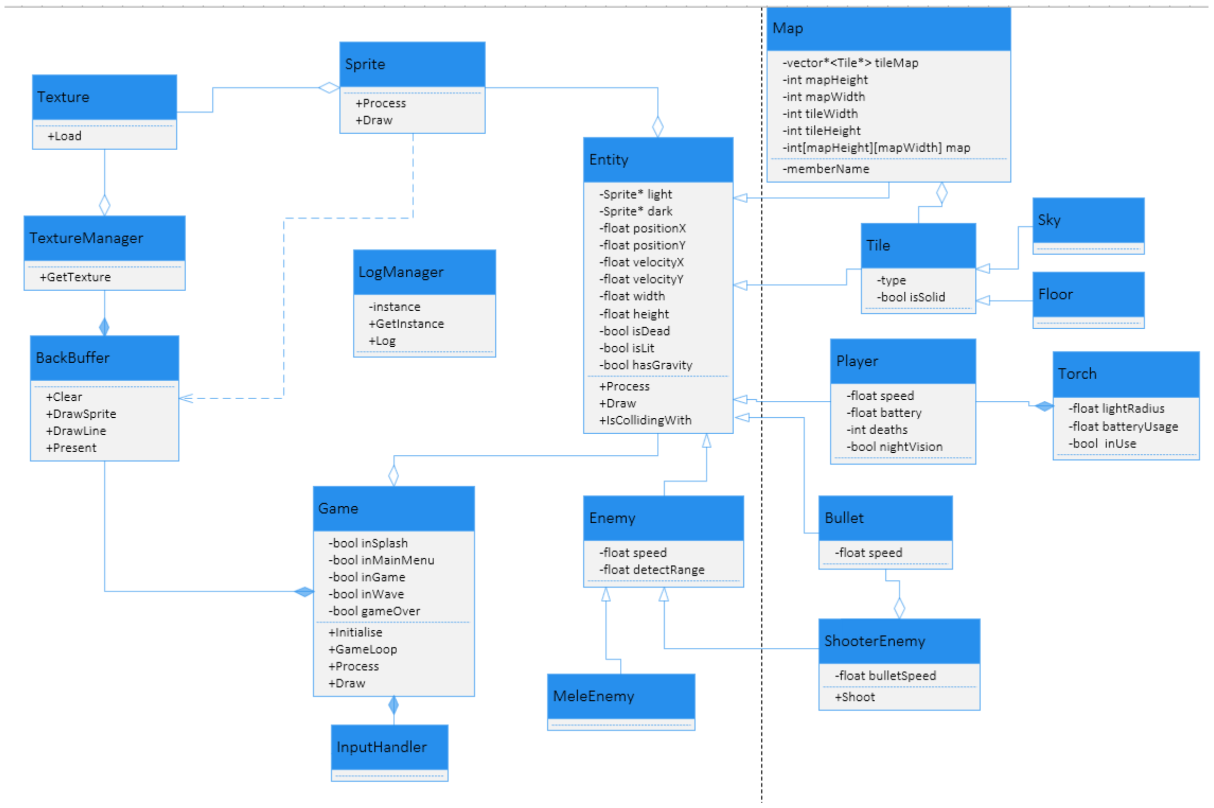
Required Technology:
Required Software e.g. SVN, Discord, visual studio enterprise 2017, etc.

Key Technical Challenges:
Animation, level design. Possible solutions are using premade animated sprites and map tile sets.

Most of us are not familiar with working with sprites and animation so we believe that animation would be a more difficult area of the games development. Especially with one of our core features being a flashlight, that will alter the visibility, sprites, or animations of the game's entities.

Architecture Overview:

Class diagram



Key Algorithms:

Enemy States: The games enemies will use a collision-based detection to switch from a passive state into an aggressive state. The enemy will switch back into a passive state after the enemy leaves the detection range (possible timer to maintain the chase state for a short duration). The aggressive and passive states will be handled differently based on the enemy attack being ranged or melee.

Flashlight: The flashlight is one of the games key features. The game will naturally be a dark theme, with limited visibility. The entities in the game will toggle between lit and unlit states. When the light is colliding with an entity, it will switch its state. The ground/scenery will simply change its lighting. While enemies will toggle between visible and invisible to the player. We currently plan for 2 main enemy types. Solid enemies will be invisible to the player while in the dark and become visible when the light is on them. Ghost type enemies will be visible in the dark and disappear when the light is shone on them.

Mele Enemy States: Mele enemies will either walk back and forward or stand idle while in their passive state. When they are in the chase state, they will path horizontally towards the players Xposition. They will be affected by gravity, and so naturally fall downwards if they walk off the edge of a platform. If the enemies collide with a wall while chasing the player, they will consistently jump, either following the player, or jumping next to the wall until the player leaves their detection range, or the player passes them, and they start moving in the other direction.

Ranged Enemy States: Ranged enemies will stand idle in their passive state. While in their aggressive state they will continuously shoot projectiles towards the player based on a timer. There is potential for multiple types of ranged enemies, adjusting the projectiles gravity, speed, directional vs aimed projectiles.

Gravity: As a platformer, the gravity is a key aspect of the game. The games entities will have a gravity variable, initialised based on the entity type. The gravity is applied as a downwards velocity every frame.

Development Methodology:

We decided to use scrum as our methodology as we are all familiar with this methodology and it would make it easier for us to focus on the game instead of learning a whole new methodology.

Risks:

Member attendance, communication, scheduling, memory leaks, unfamiliar programming environment

During week 8 we had some troubles with everyone attending the labs which may be a potential risk we need to face.

Also, most of us are not used to C++ and are not familiar with dealing with memory leaks or other things semi-unique to C++.

Another risk we may face is disruption of communication channels due to the pandemic.

Scoping:

Required features, nice to have features and optional features

Required:

Character, movement, complete map, flashlight, puzzles

Nice to have:

Enemies, multiple maps, fluid movement, good performance, difficulty settings

Optional features:

Feature Importance:

We decided that the features from most important to least important are:

Working game, Good Performance, Fluid movement, Good looking map(s), sprites and animation.

Third Party API and Middleware:

- COMP710 2D C++ Framework
- Simple DirectMedia Layer (SDL2) including SDL2_Image and SDL2_TTF APIs
- FMOD Core (Low-Level) API
- Box2D