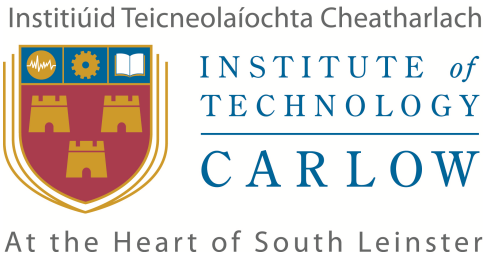

Computer Games Development CW208
Technical Design Document
Year IV

Michal Krzyzanowski
C00240696
28/04/2022

Contents

Brief description of project technicalities

Here, I'll briefly describe some of the technicalities of the project. The project is based around the physics simulation that is handled by the Box2D library, from gravity to resolving complex collisions. Box2D is relatively easy to set up but requires some tinkering when implementing functionality that directly interacts with Box2D components e.g. removing a shape after it collides with another shape that holds a custom tag. For rendering, I use SDL2, a library written in C for handling rendering, keyboard/mouse/controller events, etc. SDL2 is more hands on than Box2D since a lot of the things I wanted to do required me to code it up from scratch, such as a rotatable square shape. SDL2 comes with basic rect shapes that only have a position variable and the dimensions (width, height). Nearly everything in SDL2 is handled as a raw pointer so I had to make sure I had no memory leaks in the project. Both Box2D & SDL2 are open source libraries which helped since I needed to slightly alter some Box2D code to make it work with my rendering functionality.

The biggest part of this project, aside from the difficulty estimation, is the ConvexShape class. This class's purpose is to hold all of the Box2D shape data, since there are many different variables that you need to manage for each Box2D shape (fixture, shape, body, bodydef). This makes it easier to deallocate memory when a shape goes out of scope or needs to be destroyed. The class has a constructor that allows the user to set the initial position, shape type, etc. The ConvexShape class also has a launch() function that provides the ability to apply a force in a given direction to a shape. This is only used for "Bullet" type shapes.

Setting up the rendering was tricky since none of the base SDL2 shapes have built-in rotations such as in SFML, another rendering library. To solve this, I researched Box2D and found that it comes with a base class, b2Draw, that allows the user to implement their own rendering using SDL2, OpenGL, etc. while using internal Box2D data. I created my own class, SDLDraw, that extends b2Draw and overridden the virtual functions I needed (mainly the DrawSolidPolygon() function). I also edited the source code of b2Draw so that I could change the color of each shape based on the type (Block, Target, etc.).

Another tricky part that I spent a lot of time figuring out was implementing functionality such as shape destruction after a collision happened between two tagged shapes. I had to create a class that would inherit from b2ContactListener class in order to give me access to Box2D's collision callbacks, I called it ConvexShapeContactListener. Every time a collision happened, the functions in this class would run. I had to set up user data for each shape so that Box2D could access some of the custom functionality I have created for the shapes in my ConvexShape class such as the Type. The shapes would then react differently based on the type of the shape they collided with e.g. target with bullet collision, destroying the target. I used this later on to implement a feature where a target would get destroyed if a collision between a standard block exceeds a certain force.

Lastly, I separated the main game (placing blocks, saving level, playing level etc.) & the difficulty estimation, they are two different programs. The way the difficulty estimation

triggers is still the same (Press the ‘G’ key in the main game). The main game launches the difficulty estimation using a CMD command “Difficulty Estimation.exe sim-level”, the current leveled is saved to a level called “sim-level” and it’s passed in to the command as a parameter. Since the estimation program is similar in structure to the main game (starts in main(), that returns a value), the main game is made to wait until the difficulty estimation ends. To solve this, I used SDL2 multithreading to a basic degree. I have a vector of SDL threads that push back a new thread every time the difficulty estimation program is run.

Code: m_threads.push_back(SDL_CreateThread(runEstimation, "Estimation Thread", (void*)NULL));

CRC Cards

Class: Game

Responsibilities	**Collaborators**
- Manages the game loop. - Sets up and maintains the Box2D world. - Let the user place shapes. - Provides level saving and loading. - Handles the main rendering.	- ConvexShape - Button - Timer - SDLDraw - ConvexShapeContactListener - LevelList

Class: ConvexShape

Responsibilities	**Collaborators**
- Manages all of the necessary objects for each of the Box2D shapes. - Provides tags for the shapes. - Provides shape data that is used when saving a level.	- Game - ConvexShapeContactListener

Class: ConvexShapeContactListener

Responsibilities	**Collaborators**

- Destroys shapes tagged as Targets after certain conditions are met.	- Game - ConvexShape

Class: LevelList

Responsibilities	**Collaborators**
- Manages a list of levels grabbed dynamically from the “Levels” directory.	- Game

Class: SDLDraw

Responsibilities	**Collaborators**
- Correctly & accurately renders the Box2D shapes using SDL2	- Game

Class: Timer

Responsibilities	**Collaborators**
- Tracks time using SDL2 timers.	- Game

Class: Button

Responsibilities	**Collaborators**
- Renders a universal button shape, to be used when setting up UI.	- Game

Class Diagram