Project Wind Haven Development Journal

Sam Ward17004205

This journal details the main areas of the project I was responsible for and covers other pieces of contribution throughout the project.

Game States

I looked after implementation of the game states as I had the experience from the individual game. For the team game the states and game flow were greatly improved over my previous game, utilising a great level of inheritance with the menustate class and corresponding subclasses. The confusion from the individual game was tidied up by moving methods out of the game class to the appropriate state class e.g. directly calling a player method from the inputhandler. Resulting in removal of a lot of the static casting that I wasn't happy with in the individual game.

I introduced a states enum to better identify the states (previously just ints), and the menu states also contained their own enums for menu items.

For the menus text I wanted to create an enum or class for the text colours that can be easily referenced rather than copy paste as I have achieved this easily with java before using an enum. I did quite a bit of research into this, but it seems enums are a lot more basic in c++ than java and it's not doable, that pointed me towards creating a colours class with static members. I got as far as trying a map, but an array cannot be stored in a map so that didn't work. I then tried a colorRGB struct, which is an int[3], and then storing that in the array, could not get to work. This was all in the header file, next thing to do would be implement the colours in the cpp file instead.

Input Handler

I was in charge of the game controls; this is due to it being tied in with the game states. Much of it was similar to the individual game, however slightly improved upon. This time I utilised the state enum mentioned above for better readability, and the inputhandler maintained a reference to the game current state which made checking a lot clearer and easier than before. On top of this I designed it so some of the player methods are called directly to the player, rather than going through an intermediary method in the game class.

Player

I designed and implemented the mechanics for the player character. This was originally going to be a top down animated sprite with multi directional movement and rotation, however it ended up being a side on sprite which complicated things a bit.

Player Controls

Player controls were achieved using both analog sticks for movement and rotation, the algorithms for this weren't too complicated but was nice to get it running smoothly. Required a bit of research into the arctan function. After changing to a side on sprite the rotation factor was changed to a rotating target rather than the sprite itself, though the principle remained the same.

Grappling hook

Did not get around to creating the grappling hook. The intent was to utilise box 2d and through my research for this we ended up going with box2d for the collisions.

Collision Detection

I played a small part in the collision detection. I wanted to improve on this from my individual game so did some research into different styles of collision detection. Since this game has rectangle sprites as well as circle sprites my intention was to implement some AABB detection and circle detection, I also looked at using Separating Axis Theorem (SAT) for all collisions rather than writing different functions, I thought this coupled with broad phase/narrow phase detection will make for a versatile system. With the game using a tile system it would be easy to implement the broad phase for each entity (current tile + 8 surrounding tiles). Follow on from this I looked at the possibility of just implementing box2d and having that look after collisions, though not much use for the rest of the physics engine in this game. Ultimately the collision detection was being looked after by another team member, so I looked at using box2d in the player grapple instead.

Update

Ended up taking over this once the decision was made to move to box2d.

Box2d

Did quite extensive research into box2d, found it quite hard to get my head around. Initially this was for the grapple but in the end, it was used for collisions.

Implemented b2Draw for drawing debug.

```
class BackBuffer;


class B2DebugDraw : public b2Draw {
public:

    B2DebugDraw(BackBuffer& backBuffer)  ;


    void DrawSolidPolygon(const b2Vec2* vertices, int32 vertexCount, const b2Color& color);
    void DrawPolygon(const b2Vec2* vertices, int32 vertexCount, const b2Color& color);
    void DrawCircle(const b2Vec2& center, float radius, const b2Color& color);
    void DrawSolidCircle(const b2Vec2& center, float radius, const b2Vec2& axis, const b2Color& color);
    void DrawSegment(const b2Vec2& p1, const b2Vec2& p2, const b2Color& color);
    void DrawTransform(const b2Transform& xf);
    void DrawPoint(const b2Vec2& p, float size, const b2Color& color);

    BackBuffer* m_pBackBuffer;

};
```

When implementing box2d for the room tiles, ran into an issue where at game initiation all of the game tiles are created, so the bodies were being created for every tile rather than just the ones currently displayed ending up with many bodies overlapping (as they share same coordinates between rooms). Fixed this by resetting b2world on room transition and creating the bodies there, essentially giving each room its own world.

One of the major difficulties faced was trying to integrate box2d toward the end of t he game after most of it is built. Many issues with units mismatching etc and rewriting code. If we had of made this decision early on, I think it would have gone a lot smoother.

Debug Features

I looked after most of the debug features, as it was part of the debug interface. The final debug features differ from the originally proposed ones as we decided that some of them were unnecessary. We ended up going with four important ones being debug info display, player invincibility, increase player speed, and add key. Aaron developed the debug info display and I integrated it with the box2d debug display into the debug menu. For the other three features these were very straight forward single line functions.

Meetings

Didn't have a set meeting schedule outside of the class tutorials. During class we discussed broader things like game direction, or any issues that needed solving with the help of the lecturer. Outside of class we used a discord forum for discussions, and Trello Board for dividing up the work. Any major issues/decisions that couldn't be solved through text discussion we would have a voice chat meeting on discord.

18/10

Online meeting to assess project priorities. It became apparent that we weren't going to complete all aspects to 100%, so a discussion was had to prioritise remaining work. It was decided that less important features such as the player grapple ability and second enemy were to be dropped, and those efforts refocused towards completion of more core features such as the collision detection.

21/10

Online meeting to solve the collisions, resulting in migration to box2d.

22/10

Aaron and I got together online and spent a good 5 hours completing the final tweaks to prepare the game for submission. Integrating my box2d branch with the main source turned out to be a right mission and took upwards of 2 hours to solve.

Overview of Trello Board