[29/10/17] Ideas

Right from the start, I didn't want to do asteroids, space invaders, a catcher, or a platformer, as these felt like the obvious choices.

After many days brainstorming and talking to my peers, I finally settled on combining the top-down grid style of Pokémon and the epic fights of Dark Souls 3 to create a top-down retro boss battler.

[05/11/17] Take over lab6, use the tiles

After waiting a few weeks for the labs to progress, I feel like lab 6 has enough content for me to make a real start on my game. My lecturer, Bobby, used several cpp files in this lab that would prove useful to me. Most notably this lab contains a class called cTileMap, in here I learned how Bobby used this class to create a grid of tiles that had their own textures. This helped me create a core piece of my game. Eventually I ended up with four major classes: Tile, Level, Player, and Control. A Level contained a grid of Tiles, the Player stayed in the centre of the screen and the world moves beneath him, and Control takes all the input from SLD and makes it available throughout the program in a much more useful way. Bobby as helped me make Control static so that I could create a pointer to it from anywhere in the program, trying to pass control around as a parameter was a nightmare. As an extra touch of detail, I made a couple of the Tiles animated, this helped breath some life into an otherwise dull game.

[12/11/17] Collision detection

Now that I have a level and a player who can run around inside that level, the next obvious step is collision detection. I spent many days thinking about how I wanted to handle this and for the sake of simplicity I settled on writing some code to handle collision between two rectangles without rotation. I came up this a very elegant solution for detecting the collision:

bool Level::ColDet(SDL\_Rect a, SDL\_Rect b) {

if (a.x + a.w <= b.x) return false;  
if (b.x + b.w <= a.x) return false;  
if (a.y + a.h <= b.y) return false;  
if (b.y + b.h <= a.y) return false;

return true;

}

This worked great, but it still had to be expanded upon to prevent the player from walking through walls. This was the first major challenge of the project and it took me several days to figure it out. It seemed that every piece of code I wrote had one major flaw, such as walking through tiles if you approach on a corner, or not being unable to walk diagonally into a wall without getting stuck.

After a lot of patience, I figured it all out and had working collision detection that could be used in the future to detect projectile collisions but most importantly the player would now correctly interact with any tile tagged solid. Plus, I introduced a mechanic where tiles could be tagged slow, meaning the player would walk more slowly over water, for example. All the collision detection was handled inside Level.

[19/11/17] Merge with Lab8

Lab8 introduced two new features that were essential to my game: fonts and sounds. I had to take the code written by Bobby and merge it with my own. My game was based on Lab6, not only had a lot changed since this lab, but I had also made many changes of my own. Merging these two projects was a tedious task but it allowed me to add a working score system, background music, and sound effects to my game. This was a great opportunity to practice reading and understanding another programmer’s code.

[26/11/17] Scene manager & buttons

I noticed that one of the optional tasks for this coursework was to create a SceneManager. I decided to create one first without having any scenes, I felt trying to make scenes without a scene manager would actually be harder and messier than just creating a Scene Manager. I ended up with two new classes Scene and Scene Manager, all the menus in my game (main menu, control, credits, win, and loss) are all Scenes, the scene manager can be called to transition between them and to start or end the Level. Initially I used the keyboard to trigger the transition between scenes, but this was only for testing and the next step was to add buttons. I created a button class which consisted of two sprites, one from the text and one for the animated background, the animation would trigger on hover. The Buttons worked with Control to determine whether the user had clicked a Button which would trigger a sound effect and change the current scene.

[03/12/17] Boss and shooting patterns

It is finally time to implement our win / loss condition: defeating the boss!  
This required me to complete several things: player health, player shooting, collision detection for all the projectiles, and of course the boss.  
I started by having a static target with health, this gave me something to shoot at while I worked out the math for the projectile direction. I ended up with a projectile class with an initialize function, this function would take a starting position and a unit vector for direction.  
It was at this time in the design process I realized I was using the same functions several times through out my game. I decided to make a class called tools to contain them, the functions were: round (for float and double), random (for int float and double), and calcUnitVec which calculates the unit vector between two points.  
After I could successfully shoot the boss, reduce his health to zero, and reach the win condition, it was time to make him shoot back. This was very similar to getting the player to shoot and took no time at all since most of the code I wrote was reusable.  
I now had a working win / lose condition. The next step was to allow the player to select continue if they died, this would restore their health but leave the boss’ health untouched, this makes the game easier for testing. Also, after the player selects back from the win condition to go to the main menu, some functions are called to reset everything so that when they press start a fresh game begins. The classes player, boss, and control all have reset functions.  
To add some extra challenge and depth to the game I made the boss move up and down as well as vary his attack patterns. He normally shoots a projectile directly at the player but there is a 20% chance he’ll shoot eight projectiles in all directions. This second attack is also called if the player touches the boss, this is to deter the player from getting too close.  
The final addition to the game this week is a health bar for the boss as well as one for the player. This really brings the UI to life, there was too much plain text. I created all the stages of the health bar in photoshop and switch between them with a case statement, this was simpler than trying to stretch a bar in relation to health.

[10/12/17] Save Game

The last requirement to fulfil for my coursework is reading and writing to a file. I decided to create a save game feature. If the player presses escape while in the level, all the data for the level will be saved and the player will return to the main menu. The start button now becomes a continue button, clicking it will load the information from the save data and allow the player to continue where they left off.

[15/12/17] Extension and Final Video

We were given an extension to our coursework, I have written about what I did with this extra time in my Wiki.  
Here is a link to a gameplay video: https://youtu.be/SySAePcCIrY