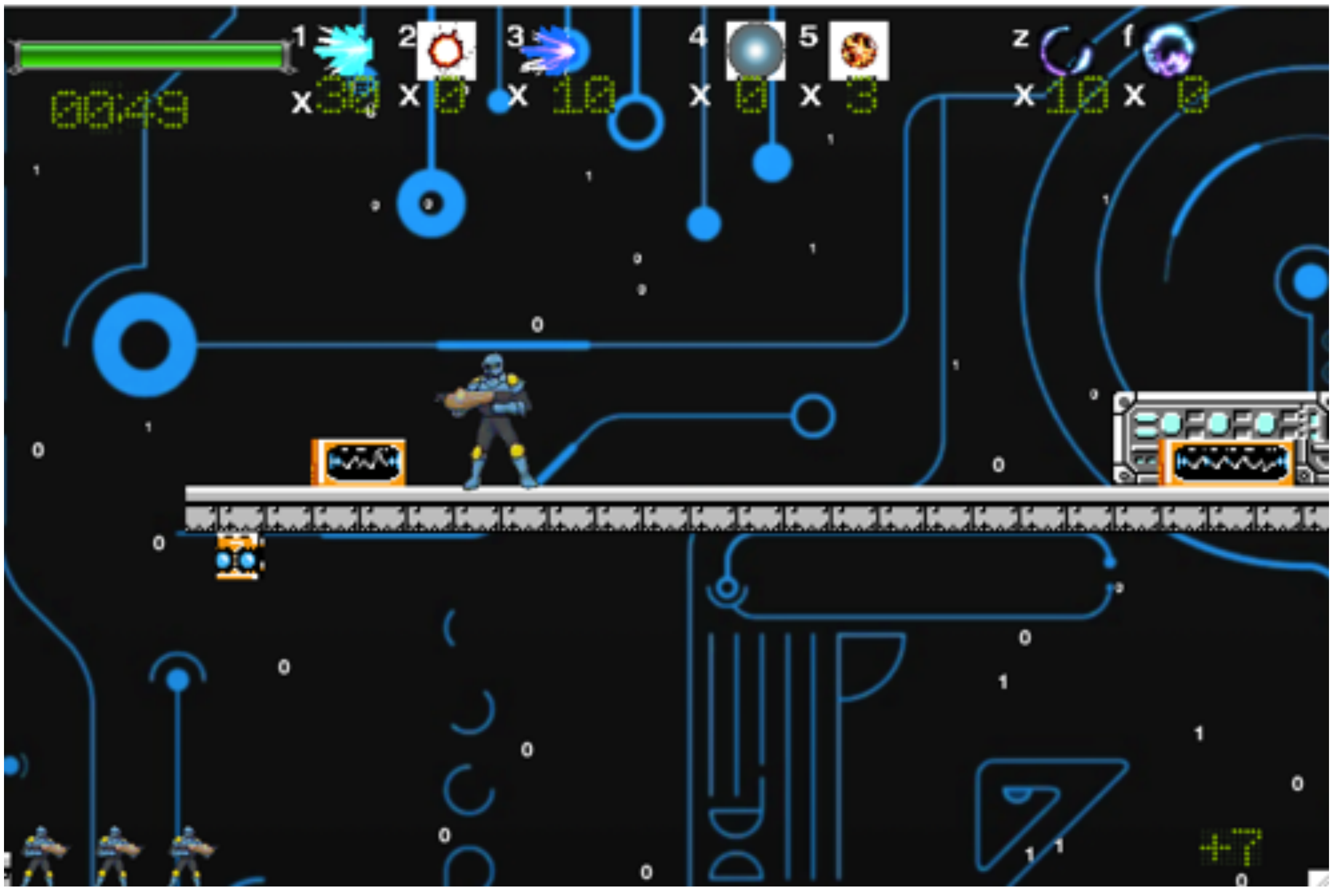
Revenge of the Code

Software Engineering - Final Project Report



CS335 Spring 2015

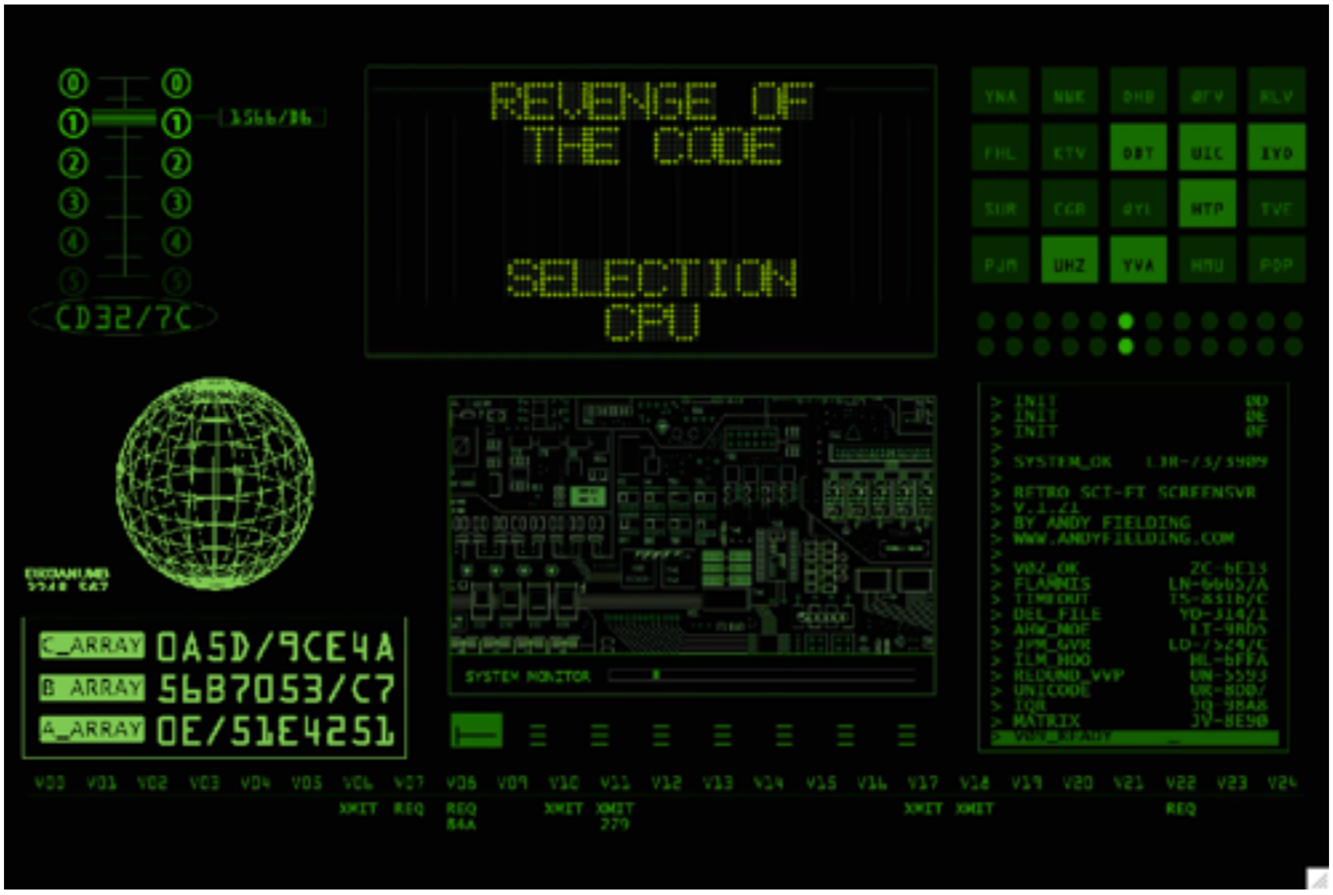
Group 8
Bethany Armitage
Chad Danner
Ted Pascua
Brian Singenstrew

Summary

The gameplay concept of Revenge of the Code centers around an anthropomorphized antivirus on a mission to debug a computer system of malicious viruses that have taken over. The game design is a two dimensional platform side scroller. The implementation of the game was programmed by coding in C++ with an object oriented design. The game design of this software engineering project displays the MVC (model, view, control) architecture used to design software.

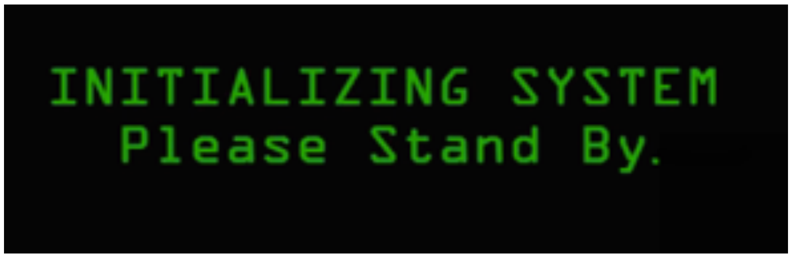
Gameplay

In order to create the game for entertainment, the idea was to keep in character as much as possible. Therefore, the game was implemented as if it is not a game, but a mission. The initial scene a user sees is a computer screen that doubles as a menu. The computer screen is animated showing several blinking lights, and scrolling text. The user can navigate with the arrow keys to different components to “debug”, or exit the system. The selected component is highlighted in light green. Upon selection of any component except RAM, the user finds that selection will not be allowed. An error message is displayed in the center console redirecting them to the RAM component. This was done to force a sequential ordering to the game. RAM is essentially level 1. The user then selects RAM by pressing the Return key.



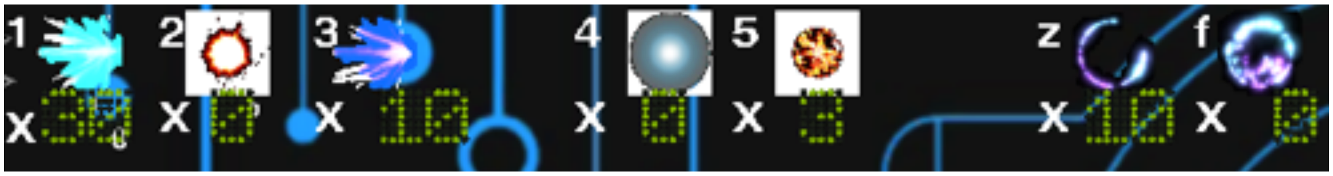
```
> INIT                          0D
> INIT                          0E
> INIT                          0F
>
> SYSTEM_OK     L3R-/3/5909
>
> RETRO SCI-FI SCREENSVR
> V.1.21
> BY ANDY FIELDING
> WWW.ANDYFIELDING.COM
>
> V0V_OK                  ZC-6E13
> FLAMNIS               LN-6665/A
> TIMEOUT               TS-8516/C
> DEL_FILE               YD-314/1
> AHW_NOF                 ET-9R05
> JPM_GVR               LD-/524/C
> ILM_HOO                 HL-6FFA
> REDUND_VVP              UN-5593
> UNICODE                 UK-8D0/
> IQR                     JQ-98A8
> MATRIX                  JV-8E90
> VVP_READY      _
```

Upon selection of RAM, The screen is shown all black where the center of the screen is shown typing out “Initializing System Please Stand By”. This functions as an aesthetic transition to the level and is not necessary for any actual loading.


INITIALIZING SYSTEM
Please Stand By.

Finally, the user is dropped in the level itself. No immediate harm presents itself to allow the user to experiment with controls and survey the screen. The user may attempt to move with arrow keys or the classic “w, a, d” keys typically used for game movement. Both sets of keys work identically for movement. Intuitively, left/a is to move left, right/d is to move right, and up/w is to jump. Space bar is for shooting, and item usage is displayed at the top of the screen in the inventory.

The user also notices a health bar and running timer on the top left, a score at the bottom right, and lives icons on the bottom left. This indicates that the gameplay entails that damage is dealt by the player, damage is taken by the player, and they have so many chances to try again. The player finds that they are not allowed to move to the left, so they begin their journey by moving right, and up if they can no longer proceed right. The user shoots any virus enemies they find along the way. They find items lying around in some places, and they find that enemies sometimes drop items upon death. The quantity of items, type of items and key to press to use each item is depicted in the inventory at the top of the screen.



Any time the user runs out of health by projectiles absorbed from enemies, they are placed back at the beginning of the level, and a life is subtracted from their current lives. When the last life icon is gone, they have one more chance to continue. Upon dying with zero life icons, they are kicked back to the menu to restart the game. When the user successfully navigates to the end of the level, they are confronted with the boss enemy virus that spawns the smaller enemy viruses. Upon successful destruction of the final boss, it explodes, the user sees the credits screens, and they are then back at the main menu. The game only has one level implemented, but had there been more, they would then be able to select the next component to debug, which would be harder than the first.

MVC Design

In order to be a quality software product, the program implemented the MVC (Model, View, Control) architecture design.The main part of the program fist set up the model for the game then continued in a loop through functions that updated the view and implemented the control.

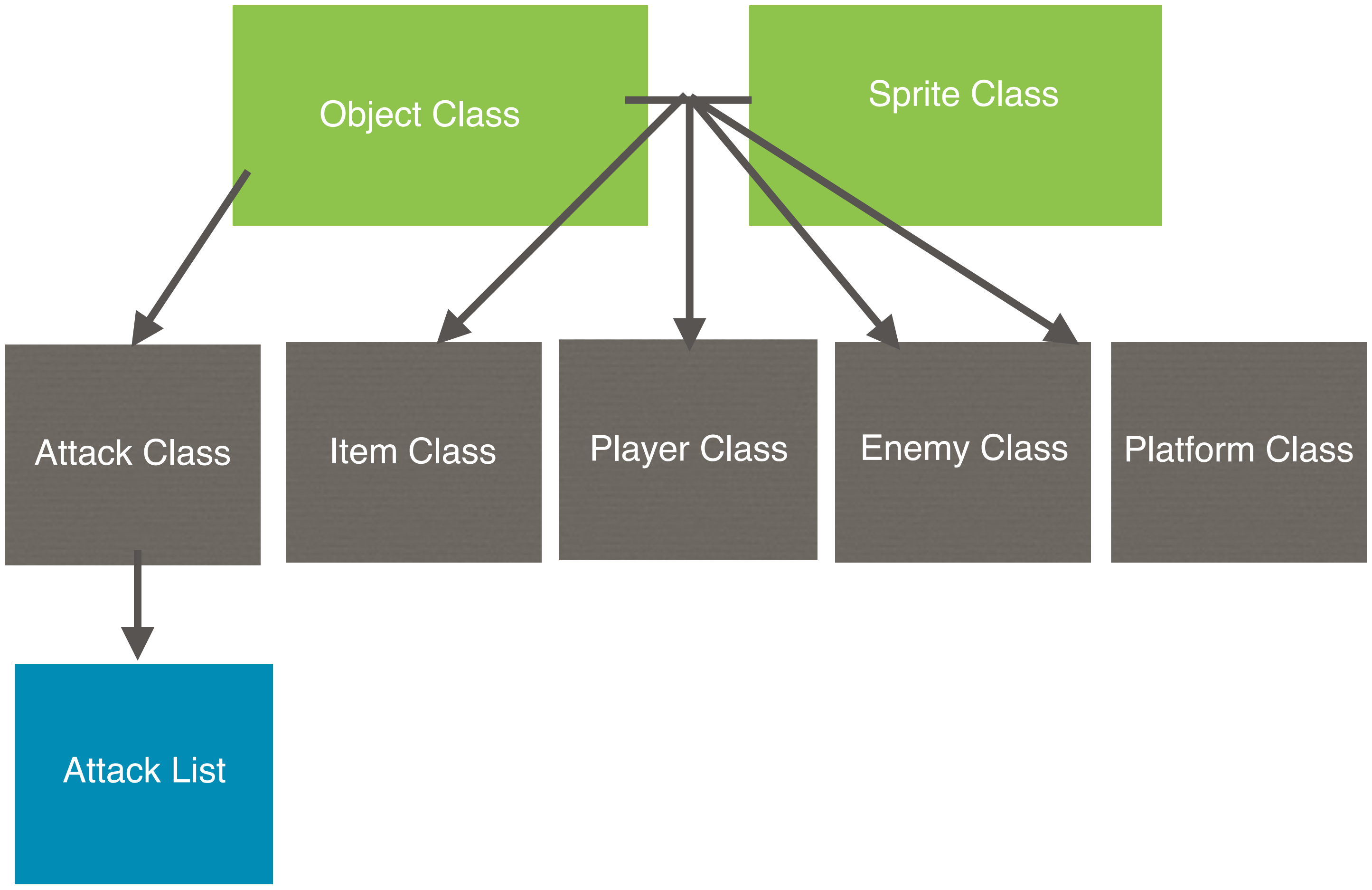
The model of the game can be described as built of specific components that related to environment the user interacts with and the statistics tracked. Simply put: there are several platforms that the user is able to interact with, there are enemies and items placed on specific platforms, and the user has a health, a score, lives, and collected items. Each of these are arbitrary to the design of the program, and can be changed independently of each gameplay. For example, the same user can play the game twice and end up with a different number of lives, scores, or items at the end of the level the second time through. The model constructs a template for each of the pieces of the game, and is manipulated as the game is being played.

What the user sees on the screen is the View component of the MVC architecture. The implementation of the view was constructed by using the Unix X Window System (X11) to contain the display, and Open Graphics Library (openGL) API to render the display. It is essential that events that occur are visually indicated in order for the game to function. For example, if the user controlling the hero player decides to run, they press the right arrow, and they see that the hero runs to the right. The user must be able to see the platforms they are running on, the enemies they encounter, and any statistics from the model. A few things they also need to track are their health level, the items they've collected and the number of enemies they've killed. Based on the state the player was in (main menu, level 1, pause menu, credits scene), the appropriate rendering function was called to display the desired graphics by using the openGL API.

The control was implemented by processing events that were captured within the X Windows display, and any other manipulation of the model. A good example of the control portion of the program is the connection between when the user hits the arrow key to run, and sees that the player is running. The running animation of cycling through the sequence of sprite images is part of the control. Also, when a player kills an enemy, the score is incremented. The control takes care these. The model is represented in the view, which is updated by the control.

Object Oriented Design

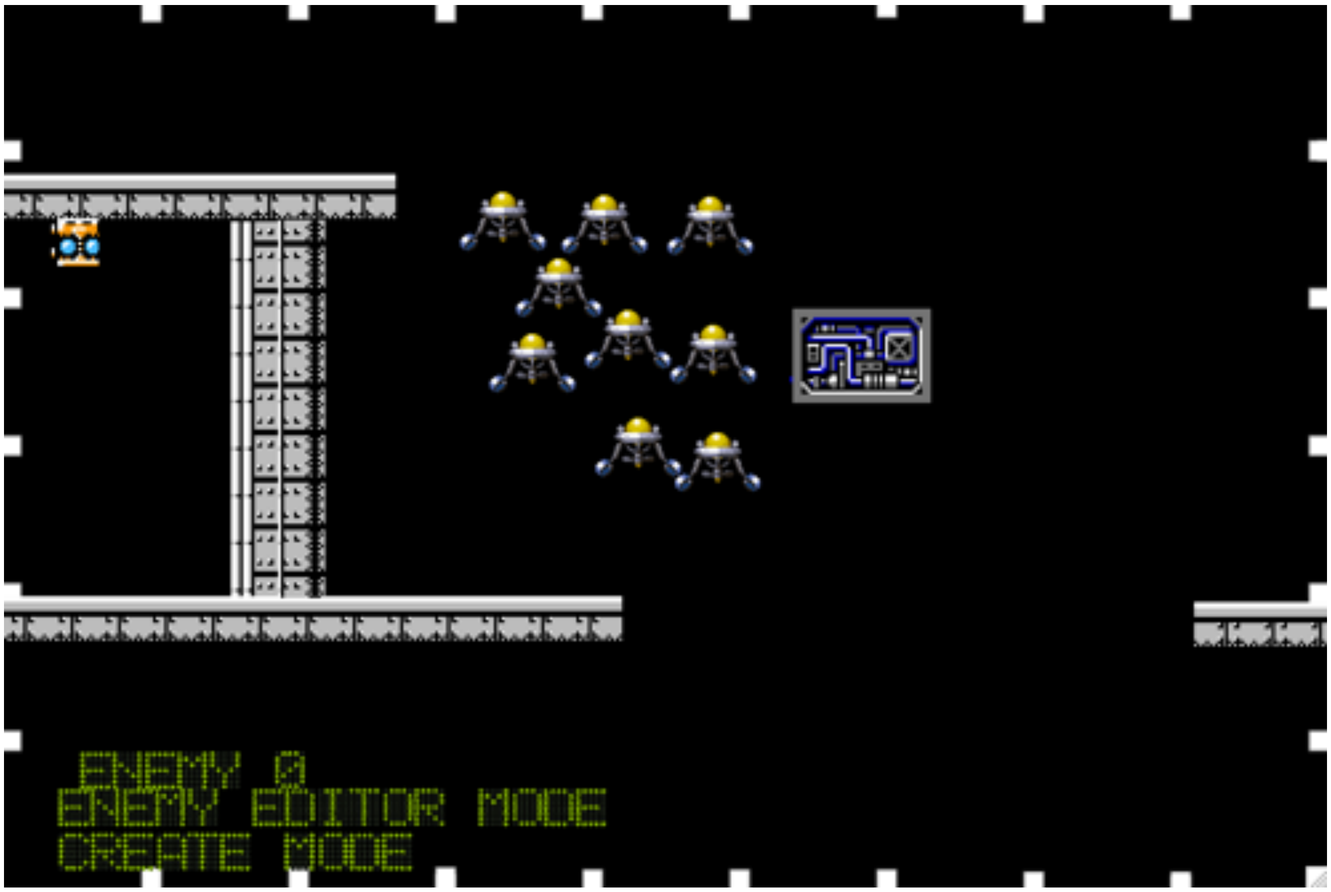
The language used to code the game was C++, which is an object oriented programming language. The basis for the design of the game started with the Object class. Most other classes inherit from the Object class. The object class design was to allow control over the creation of objects for the model. It has a height, width, center, and a few other universal properties. At the same level is the sprite class. Sprites have textures mapped to them, and may be able to move, Sprites need to be able to face left or right, and they must be able to cycle through a sequence of images to display animation. The Platform class is both an object and a sprite. It shares properties of both. It has a texture mapped to it, and a width and height, but Platforms can be extended in size with the texture multiplying instead of stretching. The Enemy class is also both A sprite and an Object. The Enemy class implements artificial intelligence so that enemies can be moving around without control by the user. They can appear to move intelligently. The Player class is also both an Object and a Sprite. The player is the opposite of the enemy. It's decisions are made by user control instead of artificial intelligence, however. The Attack class inherits from object. Different attacks can be created based on properties of the Attack class. A separate attack list class was created to quickly contain and construct several unique attacks. The Item class is both a sprite and an object, it connects the effect of a item to a player.



Highlighted Features and Designs

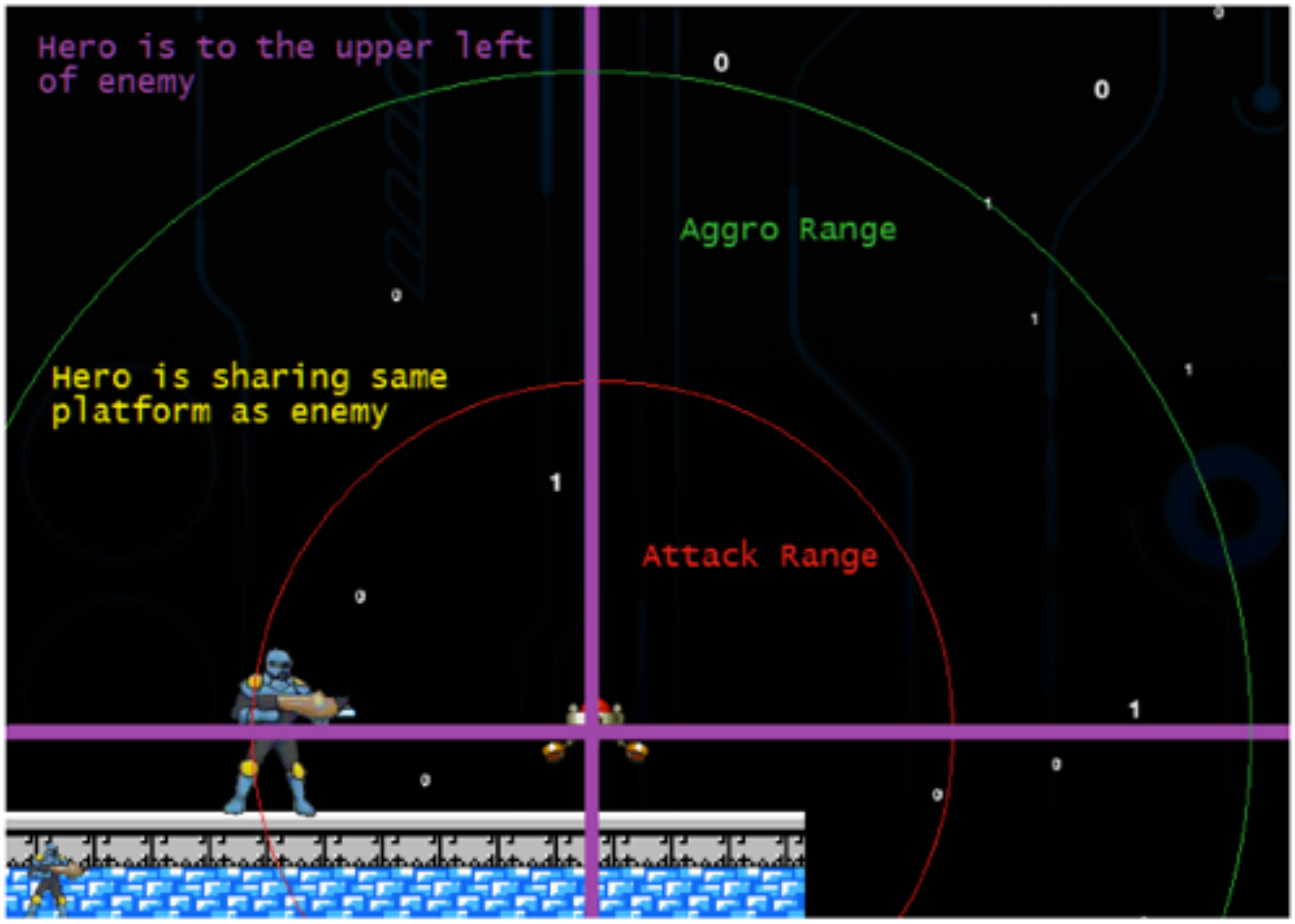
Editor Tool

One of the strategies to efficiently design the game was the creation of a level editor that would allow design of the level to be implemented in a “point and click” environment. It eliminated the task of manually hard coding each object into the game and calculating coordinates of the relationship of all the objects. The editor displayed the type of object being placed, and recorded the coordinates of the placement. When objects are placed as desired, the file is saved as a binary file and it implements object serialization.



Collisions, Physics, and AI

Collision detection is calculated at each frame between enemies and platforms, hero and platforms, hero and items, bullets and enemies, bullets and hero, and bullets and platforms. After setting the moving object outside of the collision zone, in some situations we also had to modify the moving object's velocity. In the case of the bullets, the hero and/or enemy will take damage based on a damage value in the structure of the bullet, then the bullet is deleted out of the linked list. Gravity was modified throughout the game to provide a relatively realistic look, but still have little enough gravity to jump to a platform easily. Also, movement speed of the hero and delay between animation frames was modified to have the hero's feet contact the platform in a way that looks as realistic as possible. Enemy movement speed was also modified throughout development to make the game not too easy but not too difficult. AI calculation is performed on each instance of each enemy, and outlines the behavior of the enemy. There are 3 enemy types, 3 separate defined ranges of distance between enemy and hero, and 4 different quadrants around each enemy. Based on the values of each of these specific cases, the enemy will either patrol the area, stand still, follow the hero, quit following the hero, or attack the hero.


Hero is to the upper left of enemy
Aggro Range
Hero is sharing same platform as enemy
Attack Range

Strategic Level Design

The design of the level was created to be interesting to the user, and different levels of interest can be generated by strategically designing the level. For example, if there is an item that the user desires to pick up, it can be placed in view of the user, but only reachable by following a certain path to the item. The user also must feel like they are not moving forward, but exploring and navigating. Occasionally they must be forced to redirect from the level path and climb upwards to advance through the level. there are also side paths that can be taken if the user wishes to pursue exploration more than advancement.

The Attack Class

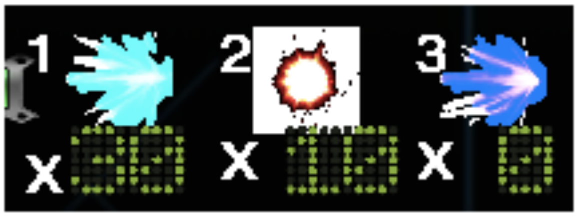
The purpose of this class is to make an object that interacts with other objects and cause certain effects. Some of those certain effects can change health, velocity, and placement of enemies or the player. The Attack Class contains functions that can change the behavior of objects such as changing speed, pushing back the caster of the attack, push other objects, change how long the object last, and much more. Here's an example below how to change the behavior.

```
attacks[id].init(width,height,0,0);
    attacks[id].changeRate(15);
    attacks[id].setVelocityX(7);
    attacks[id].setPushBack(false);
    attacks[id].setPushAway(false);
    attacks[id].setTimeBase(true);
    attacks[id].setDuration(3000);
    attacks[id].setCycleBase(false);
    attacks[id].setCharges(5);
    attacks[id].setDamage(6);
```

In main.cpp while the attacks are place in, the attacks will call the function called autoset() and autoState() which change placement base on its velocity and detects if it reaches the end of time duration.

Currently the Attack Class is only inherits an object, it does not contain the Sprite Class. The reason for this is that it takes long to clone a Sprite Class Variables because of how much bytes the images uses. The Attack Class instead takes a Sprite Object as a parameter. (The reason for this is the Gluint won't work inside some OpenGL functions). This may not look optimal, but that's where the Attack-List Class comes In. (The Drawing Process will be explain in the Attack List Section).

The Attack List Class



The AttackList purpose is to organize and create preset Attacks. The other purpose of it is to have the Attack objects have the same space as the Sprite Objects Array so that the Attacks objects can easily reference to the sprites.

```
struct attack_list{
    Sprite  sprite_sheet[MAX_ATTACKS];
    Attack attacks[MAX_ATTACKS];
    int attacks_length;

    Attack *currents[MAX_CURRENTS];
    int currents_length;
};
```

Using Sprites

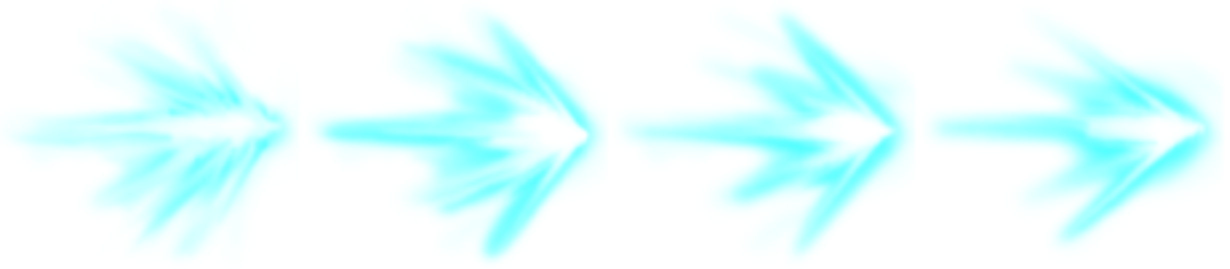
For the Attacks to reference a specific SpriteSheet, the attack object uses these functions

```
sprite_sheet[id].insert("./images/dashShield.ppm", 5, 6);
sprite_sheet[id].setSize(width,height);
attacks[id].referenceTo(sprite_sheet[id], id);
```

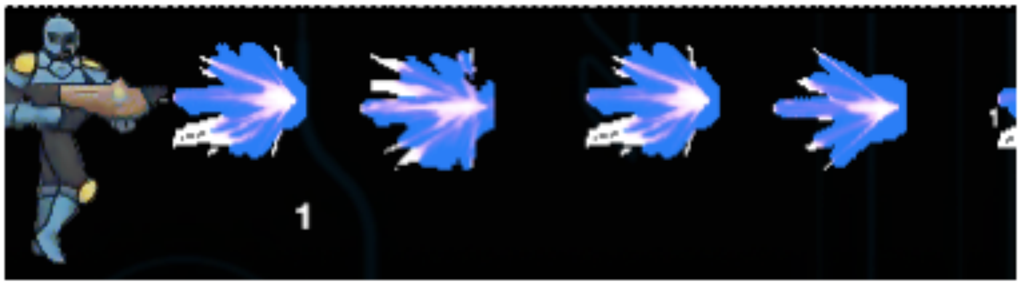
the variable 'id' is what synchronize the attacks[id] and spriteSheet[id]. Its also used to differentiate each preset attacks.
When it draws the attack, it can reference to the sprite sheet with the same id.

Animation

Each Attack has an index it keeps track of. That index will keep cycling from 0 to the the amount of tiles there are. For example, shown below are 4 tiles. When the Sprite Object is constructed, the user specifies the number of rows and columns there are. Here is one row and four columns.
If you keep cycling through these you get animated attacks.



shown in game



Making Attacks

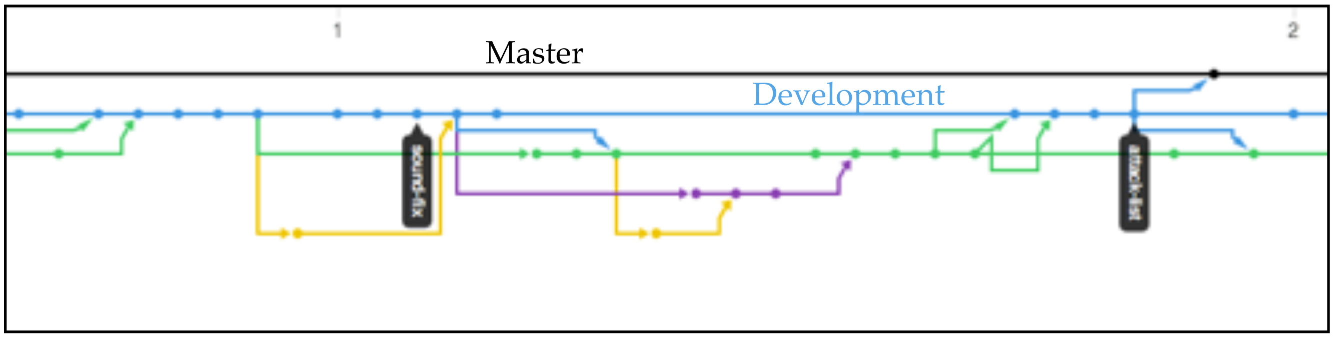
When you make an unique attack object you save it into a storage. If you want to use that attack you copy the attack object in storage. In this way you don't change the essence of the original attack, you just copy that certain attack and with different placement and some changes. You can also make massive amounts of copies of it and all of them with different position.

```
currents[currents_length] = new Attack(attacks[tId]);
currents[currents_length]->setID(currents_length);
   currents[currents_length]->targetAt(caster);
   currents[currents_length]->setCenter(caster->getCenterX(), caster->getCenterY());
currents_length++;
```

This copied attack is called currents[i] in the code and it will continuously call autoState() and autoSet() in main.cpp to move based on velocity until its self timer is exceeded. During its time existing, if it interacts with an object it's set to affect, it will cause an effect to the set object its colliding to, such as damage and push-backs.

Version Control

Git and github were used as version control for this project. This allowed quick easy integration of individual work into the main project. The strategy was to take out a development branch and keep it integrated with the master branch at several stable points. From the development branch, other branches are taken out corresponding to the feature or bug that they implement or fix. As you can see, master is mostly untouched excepted merges from development, and many branches are taken out of and merged back into development.


1
2
Master
Development
sound-fix
track-list