Take It Back!

Jacob Thurman

CIS226-HYB1

**Game Summary:**

This game will be a side-scrolling runner game. The game will have a goal taking your player to an endpoint while avoiding various obstacles, holes, weapons and baddies looking to get in the way. There will be a character traveling around a map looking for some end goal. The weapons/baddies difficulty will depend on the level. Some harder levels may have more than one type of enemy. The first few will have none. Some levels will be locked until previous levels are completed. The game will keep track of which levels you’ve completed and failed and show you that from the level picker screen.

**Project Background:**

As I thought about it, I’ve always been the most interested in games I can play quick and be done with, such as the [chrome offline dinosaur game](http://www.omgchrome.com/chrome-easter-egg-trex-game-offline/), in addition to Super Mario, another side-scroller with a goal of reaching a flag at the end. This *Take It Back* game is semi-inspired by that and somewhat so by other similar side-scrollers. I feel this game will capture all of my interests in gaming well. I feel that as I develop it, it will evolve into something more interesting and unique. This game could be a fun way to forget about stress and worry for a just a few minutes in the life of someone even busier than I. The societal impact of this game will be a positive effect on the mood of the player, the game will be designed with the intention of being easy enough not to frustrate too much and hard enough to suck the player in. I get bored very quickly of games with no goals as well, and that is how I came up with the idea of taking an object back. A quick goal to waste time, but feel accomplished in it.

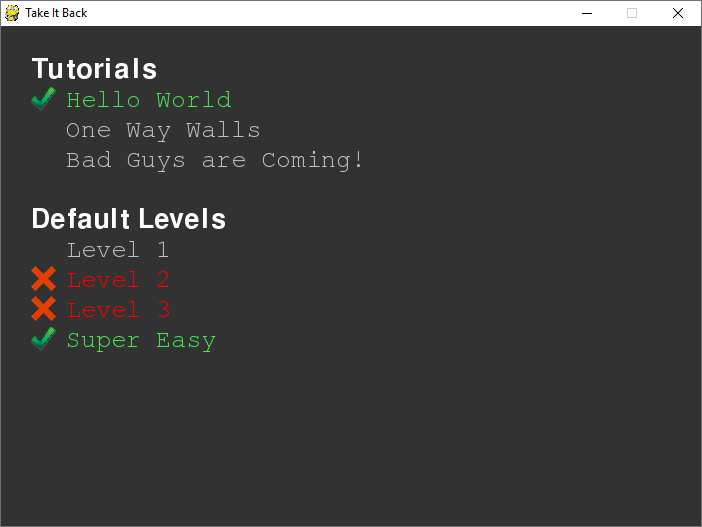
**IPOS Requirements:**

The program will receive input through keyboard interaction. The only final decision of the mapping thereof is that the spacebar will be used to jump. Whatever key combinations are used will be generic and potentially configurable in the game, depending on if I have enough time. The processing of the program will be the generation of the map and the calculation of the “camera” and such. The output will be the rendered map: the obstacles, character, enemies and such. The game will have a data.json file (format may differ) to store things like the completed and failed levels along with level packages.

**Mockups:**

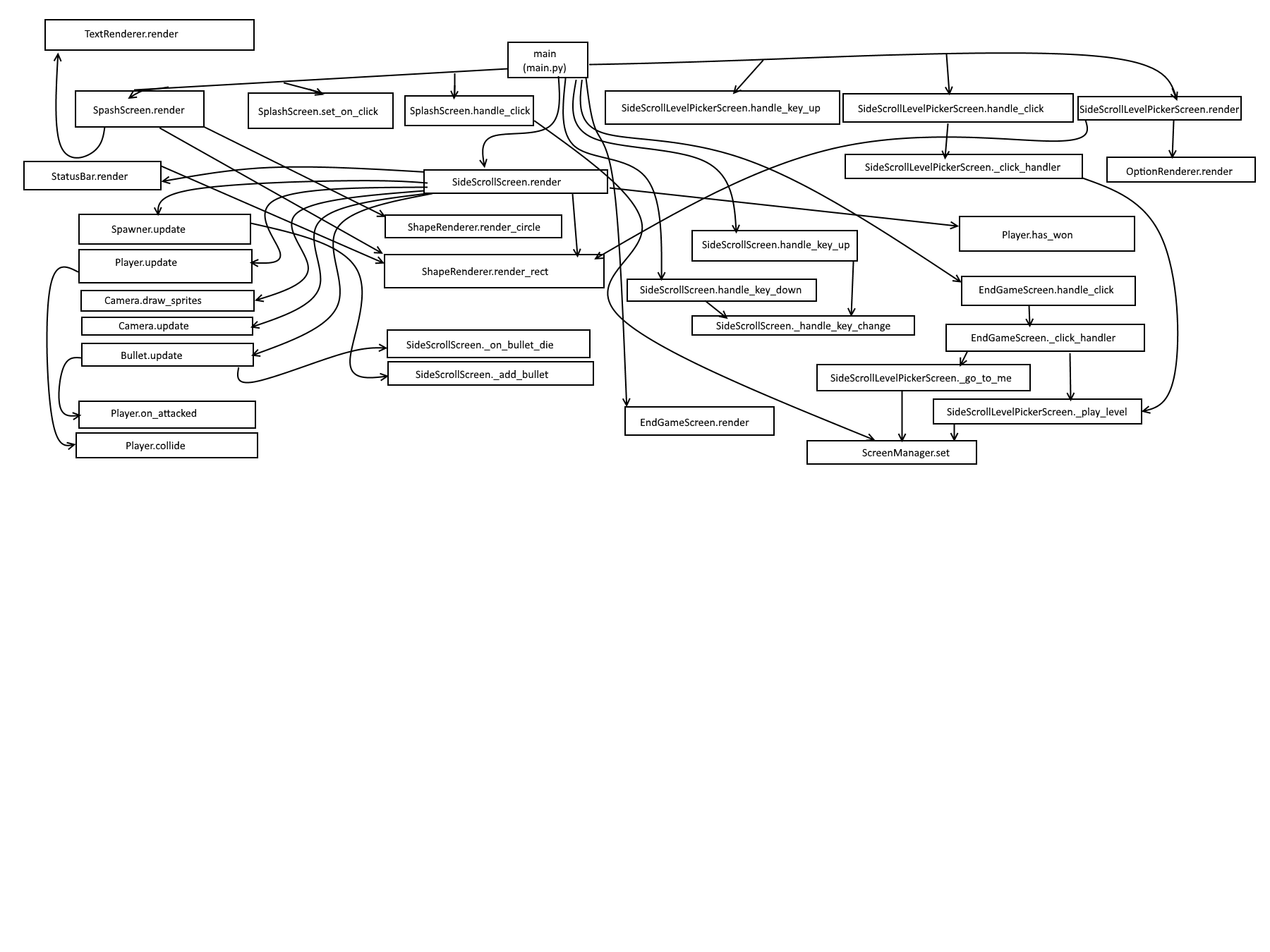
Levels are still the same as Assignment 3 just with lots of bug fixes (like fixing line of sight calculation for weapons, and fixing weapons firing to the right). I also added a limit to the weapon’s firing speed, limited bullet lifetime, and added a you-win/lose screen. Finally, I added a health bar always fixed to the bottom of the screen. It is green until your health goes below two thirds of its total value at which point it is yellow until it reaches one fifth and becomes red.

This is the level screen. This shows all of the available levels to play. This is the final version of this screen at the moment. Changes from the assignment 3 are these: Changed colors, actually added the check icon and I did add the failed level coloring I was considering.



**Structure Chart**

Finally, this is my arrow chart required. I did not include constructor calls for simplicity. (Also I didn’t know how to include that as the origin had no classes). In case this image is too small, the full .png file is located at /dev/arrow\_map.png

****

**File / Record Layout**

There are a few different pieces of data stored to make everything work for the game. The first file, “levels/data.json” is a json file containing 3 sets of information, completed levels, failed levels, and level packages. Failed and completed levels are stored as file paths. (i.e. "levels/default/level3.map".) The packages are stored as simple strings for names of sub folders of levels where there is a package, such as "tutorials" to add the package file in “levels/turorials/package.json”. This is the json structure for data.json:

{

“failed”: [ /\*map file names\*/ ],

“completed”: [ /\*map file names\*/ ],

“packages”: [ /\*package folder names\*/ ]

}

Now, each level package also has a package.json file as I briefly mentioned previously. The object in this file contain to properties: “name”, the title of the package and “levels”, an array of level objects. These level objects hold actual levels in the package. They have two properties as well: “name”, the title of the level and “map”, a relative path to the map file. Here is an example:

{

"name": "A Level Package",

"levels": [

{ "name": "Hello World", "map": "hello-world.map" },

{ "name": "Bad Guys are Coming!", "map": "bad-guys.map" }

]

}

Finally, the map files contain the layout for the world of a level. Using a few special characters, you can simply create a level in notepad (or other monospaced text editor). The different character’s meaning are this: “X” is a wall, “U” is a wall you may only pass through while traveling up, “D” is a wall you may only pass through while traveling down, “R” is a wall you may only pass through while traveling right, “L” is a wall you may only pass through while traveling left, “P” is the location of the player’s head. “W” is a win-block (touching this makes the player win) and a number creates a weapon of that power. Here is an example:

XXXXXXXXXXXXXXXXXXXXXXXXXX

W X X

W X P X

W X X

W X X

W X X

W X X

W X X

XUUUUUUUX X

X X X

X X X

X X

X X

X X

X X

X 6X

XXXXXXXXXXXXXXXXXXXXXXXXXX

**Conclusion:**

I believe that I should be approved to make this game because out of all of the options I brainstormed it was the only one that solidified in my mind as a “cool” idea. It was an idea that I felt excited to work on; something I will be proud of. I feel that this game will have a positive impact on players because it will be care free, goal oriented and pure, wholesome fun.

**References**

Carlisle, R. P. (2009). Encyclopedia of play in today's society (Vol. 1). Los Angeles: SAGE.

**Glossary**

Side-Scroller: A game that moves on to the right as the player moves

Endpoint**:** Where the level comes to a close. A point obviously marked to the user as the finish.