Name: Daniel Jaffe
Name: Tyler Hackett
Name: Keiran Glynn
Name: Brenden Stevens

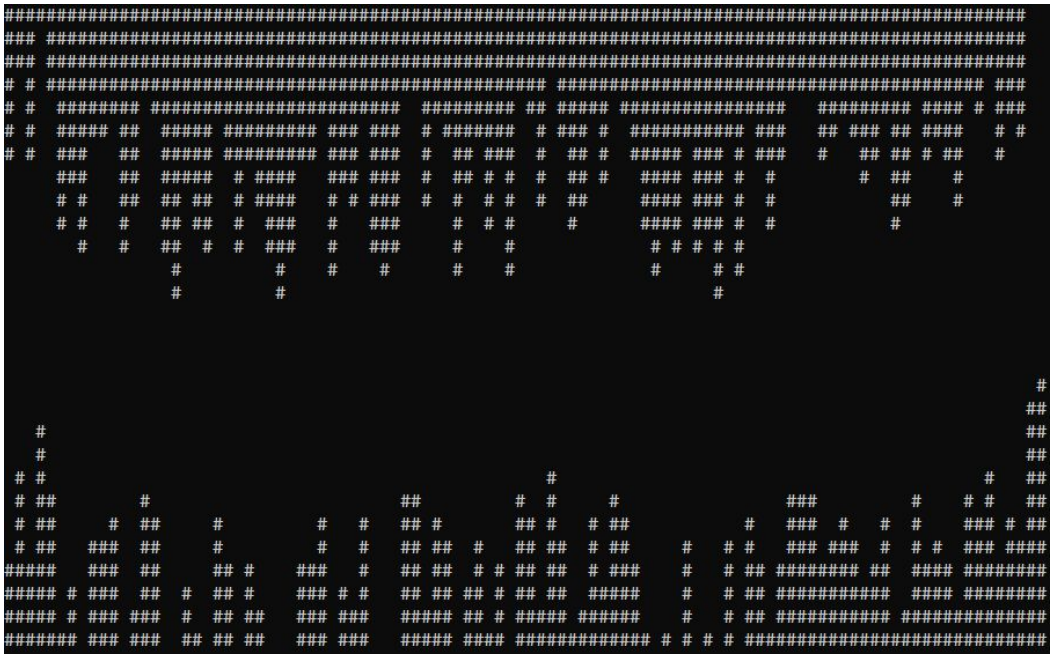
Transfer Bridge CS - Technical Project Final Report

Team Name: Moon Bears
Game: SpaceCraze
Git Repo: https://github.com/flarros/TransferBridgeFinalProject

Game Description: **SpaceCraze** is a side-scrolling action/shooter that is intended to play somewhat like space invaders but horizontally instead. As the map continuously scrolls left, the player moves to avoid walls, enemies, and enemy fire that are procedurally generating within a certain range. The player can also move up, down, forward, and backward. There are two types of enemies that have been implemented: Gunners and Kamikaze Fighters. The Gunner enemy can shoot bullets and the Kamikaze enemy will attempt to ram the player. If the player comes in contact with an enemy or a bullet the player loses health. If the player collides with a wall, the player dies. Points are awarded to the player for successfully attacking and destroying enemies. Your score is displayed at the bottom of the screen.

Game Objective: Survive and achieve the highest score possible.
Implementation Description:

Continuous Procedural Map Generation (Side Scrolling) shooter



- Fixed framerate/update rate (may run differently on each computer)
- Nearly everything inherits from an abstract base class called GameObject
- Player inherits from Actor and Actor and Item inherit from GameObject
- Game *ptr passed into GameObject allows the use of the map stored as char[][] member variable in Game class (separate from GameObject)

// ADD STUFF HERE

GameObject Class:
- The abstract base class that acts as a parent for all actors and items

Game Class:
- Contains Actors and Items as member variables
- Responsible for main game loop

Map Class:
- Generates new columns for each call of the scroll() function

Items Class:
- Items that will show up on the map
- Contain an Effect that will pass to the player when the player collects the item

Effect Class:
- Two effects implemented
    - Can heal the player
    - Can give the player triple shot

Player Class:
- Inherits Actor

Enemy Class:
- Inherits Actor

Kamikaze Class:
- Inherits Enemy

Gunner Class:
- Inherits Enemy

Individual Member Contributions:

Tyler Hackett:

I worked on much of the Game class framework and helper functions for other team members to use. During the design phase, I proposed the composition design pattern that we ultimately adopted for handling player effect items.

Kieran Glynn:

I worked with Daniel to design the inheritance relationships of our GameObjects, and organized those designs into an UML diagram. I created most of our GameObject class declarations (.h files), and implemented some of the Item and Actor classes (namely Enemy.cpp, Kamikazi.cpp, Gunner.cpp, Effect.cpp, TrippleShotPowerUp.cpp, and HealthPowerUp.cpp).

Brenden Stevens:

Worked alongside Tyler on the map generation and set up primary game loop timer. Checked logic on game member variables and passing them. Developed skeleton for final report.

Daniel Jaffe:

I worked heavily on the UML collaboration with the other team members to plan class and function implementation. I also set up our project Git repo and defined our workflow to utilize GitFlow. Beyond that, I set up the Bullet, Position, and Actor classes. I set up health checks for characters in the game, set up player movement, set up the scoreboard, and implemented checks for collisions with walls. Additionally I contributed many hotfixes to other classes and worked collaboratively on the logical structure of code and resolution of bugs.

Setbacks/Accomplishments/Future Work:

Given the time constraints it is pretty obvious what our primary limiting factor was. Simply having enough time to code, and debug the code prohibited us from accomplishing all of the functionality that we had planned on. However, this is also definitely a planning issue on behalf of the team. Had we had slightly less ambition, we may have actually been able to implement all that we planned.

Setbacks:

- Lots of valuable time was spent correcting merge issues with Github/Gitkraken, however, once it was fixed, Gitkraken may have helped speed development along
- As a team, we could not agree on the mapping method (the way we would manipulate the map), until the end of the first day
- Often times whiles coding, we found ourselves waiting on others to finish a piece of code needed to implement our own part correctly, though this is likely common

in projects of this nature, it is very possible that some forethought on the coding conventions and better delegation could have prevented some of this error

Completed:

- Getting a player to spawn
- Getting an enemy to spawn
- Scrolling the map
- Moving the player
- Having an extensive and abstract inheritance structure that, when implemented, allowed for the near instantaneous creation of a group of enemies alongside the player and power up items that followed all of the game rules
- Every class is abstracted in such a way that the project is easily expandable and even possible able to be copied over to an engine that actually used graphics

Future Work:

- Adding a boss
- Correctly implementing the power-ups
- Adding and taking away active effects from players and enemies
- Changing the type of map
- Increasing the speed and level of difficulty as the game progresses

Conclusion:

In summary, we were too ambitious in the planning of our project. We were only able to accomplish about half of the planned power-ups, items, enemies, etc.
However, the classes are set up in such a way that adding new power-ups, enemies, and other functionalities would be extremely simple beyond the achieved implementation. All things considered, we think we did good for basically only one night and one day of coding.