COMP 55
DIG: System Design

Submission Date: **10/20/2019**

Names:
-James Nicholas
-Ian Hanf
-Daniel Adler
-Brandon Raboy

Report Leader: Brandon Raboy

Overall **contribution**

Ian Hanf:

UML Class Diagram, Class Descriptions, Interaction Diagram, Algorithms and Data Structures,User Interface Design and Implementation, System Architecture and System Design, Table of Contents, References

Daniel Adler:

UML Class Diagram, Class Descriptions

James Nicholas:

Class Descriptions

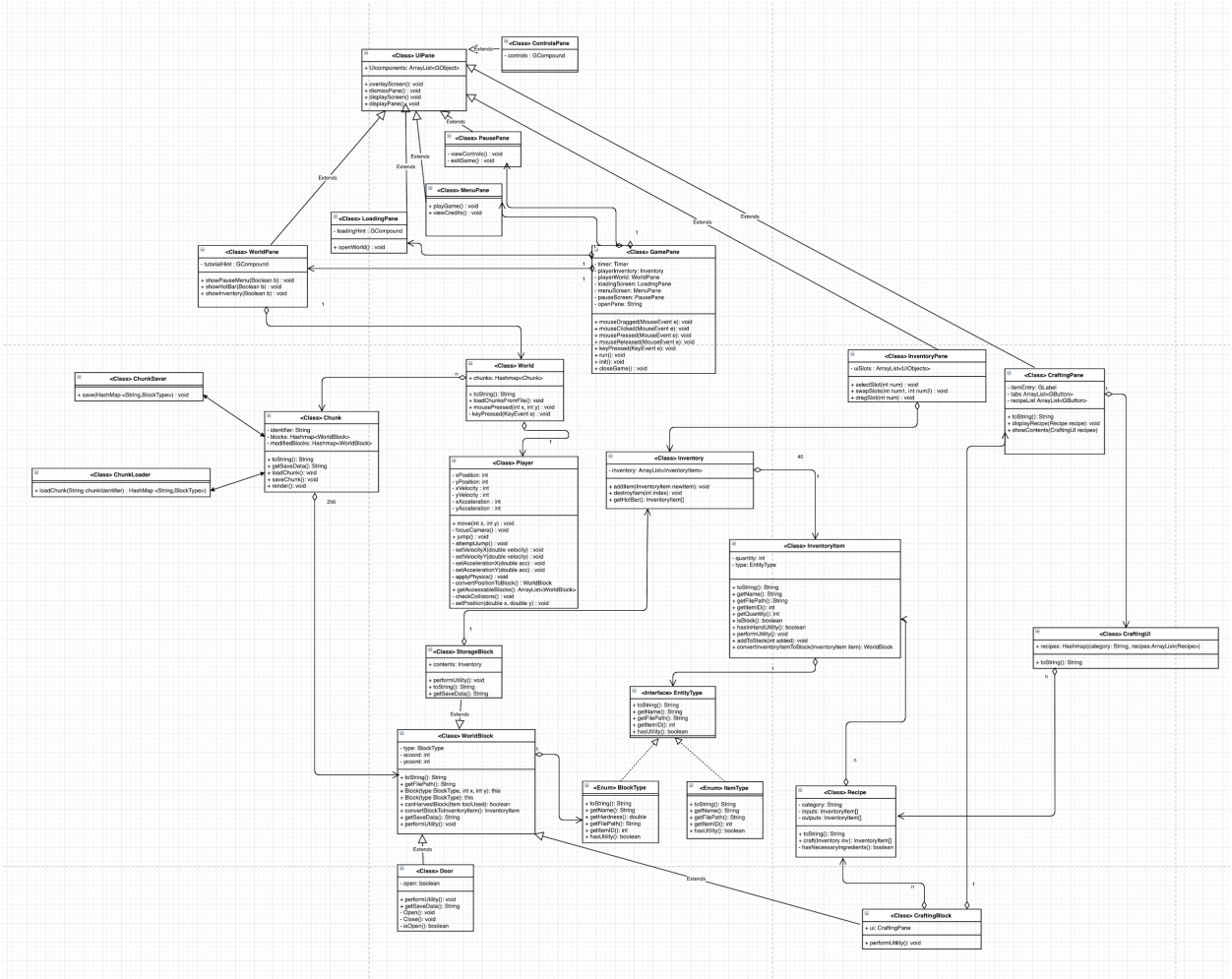
Brandon Raboy:

Table of Contents, System Architecture and System Design, References

Table of Contents

UML Class Diagram

Class Diagram



Class Descriptions

Class Game (root class)

Has a GamePane

Class UIPane

Has an arraylist of GObjects

Has a method called dismissPane, which will make UIPane null, and won't display anymore

Has a method called displayScreen, that will make UIPane visible and presented to user

Class MenuPane

Is a UIPane

Has method playGame, which will present the loading screen, and then the world that the player can play the game in

Has a viewCredits method, which will call displayPane() on the CreditsPane

Class PauseMenu

Is a UIPane

Has a viewControls method, which will call displayPane() on the ControlsPane

Has a exitGame method, which will close out the game

Class CraftingPane

Is a UIPane

Has an itemEntry GLabel

Has a tabs which is an ArrayList of GButtons

Has a recipeList which is an ArrayList of GButtons

Has a toString method

Has a displayRecipe method that will display when given a Recipe

Has a showContents method that will show the recipes of the given CraftingUI

Class ControlsPane

Is a UIPane

Has a GCompound that displays the controls

Class LoadingScreen

Is a UIScreen that displays a hint on what to do when world loads

Class WorldScreen

Is a UIScreen that embeds the World inside, and overlays the hotbar

Class Player

Has a GCompound

Class Camera

Class World

Has a Hashmap Chunks.

Has a Player

Has a Camera

Class Chunk

Has a chunkIdentifier - Example -> “0,0”

Has a Hashmap of Blocks named blocks. Unable to alter.

Has a Hashmap of Blocks named modifiedBlocks that will add keys for blocks that are placed or deleted.

Chunk has a SaveChunk helper class

Chunk has a LoadChunk helper class

If Chunks identifier is “0,0”, then it is the default loading chunk, and we will override the hashmap of blocks with preset starting area.

Chunks are 16 Blocks high and 16 Blocks wide

Chunk has a render function that will get all the Blocks from blocks and get all the modifiedBlocks, and render the blocks as a GCompound. modifiedBlocks override keys from blocks in rendering.

Class LoadChunk

LoadChunk tries to load data that has been saved for a chunk

LoadChunk has a public method called load(String chunkIdentifier) that will return a Hashmap of modifiedBlocks

The load method works by trying to find a file existing with the name of chunkIdentifier in the format of “0_0_blockData”

Class SaveChunk

Save chunk tries to represent the data behind the chunks

SaveChunk has a public method called save(Hashmap modifiedBlocks) that will save the blocks that have been changed by the player

The save method will store each chunk as a txt file using the chunk identifier: Example “0_0_blockData”

Class UIObject

Has a GCompound

Class Inventory

Is a UIObject

Class UIButton

Extends UIObject

Class Recipe

Has a HashMap named Ingredients of keys representing ingredients, and values representing the number needed

Enum outputItem

Int numberOfOutput

Enum BlockType

Enum ItemType

Interface EntityType (contains BlockType)

Class InventoryItem

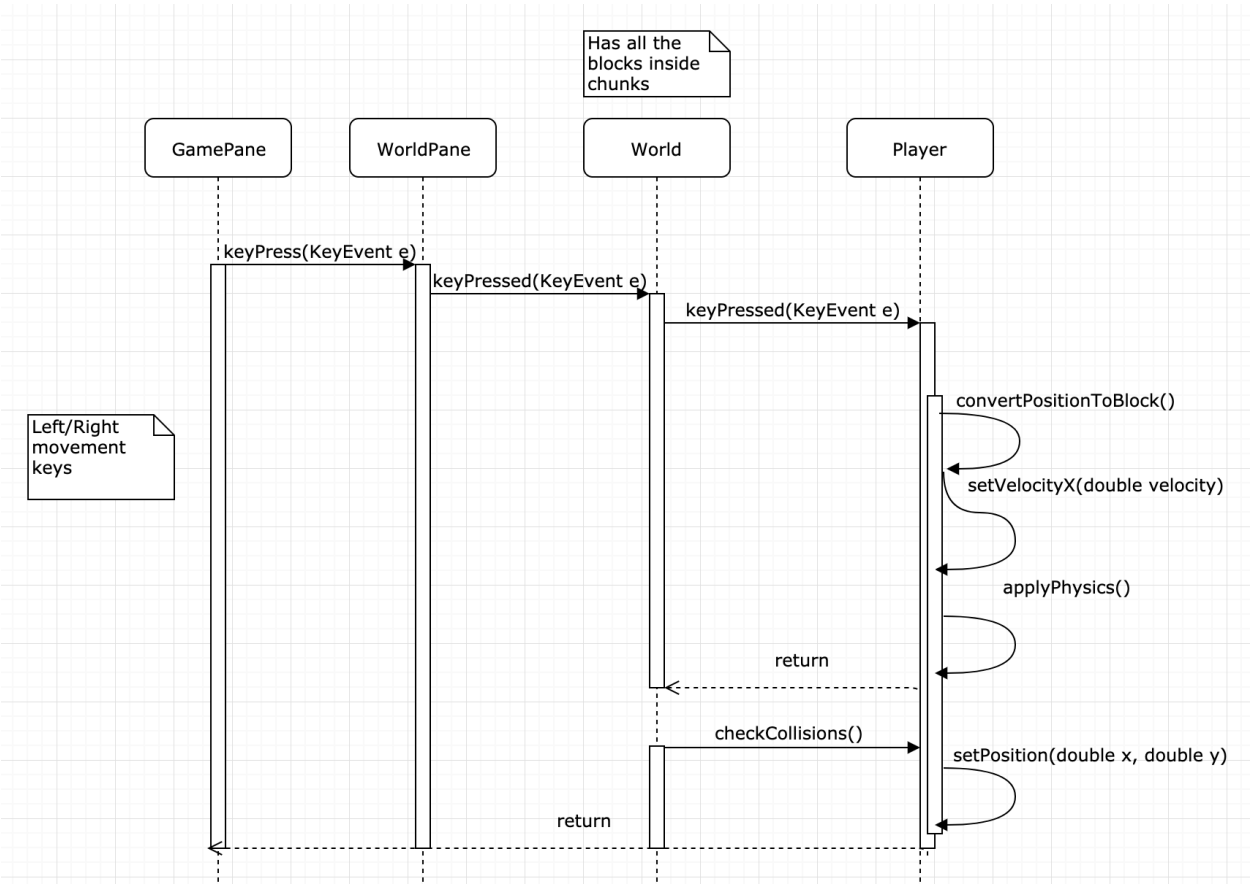
(represents an item or a block, show picture)

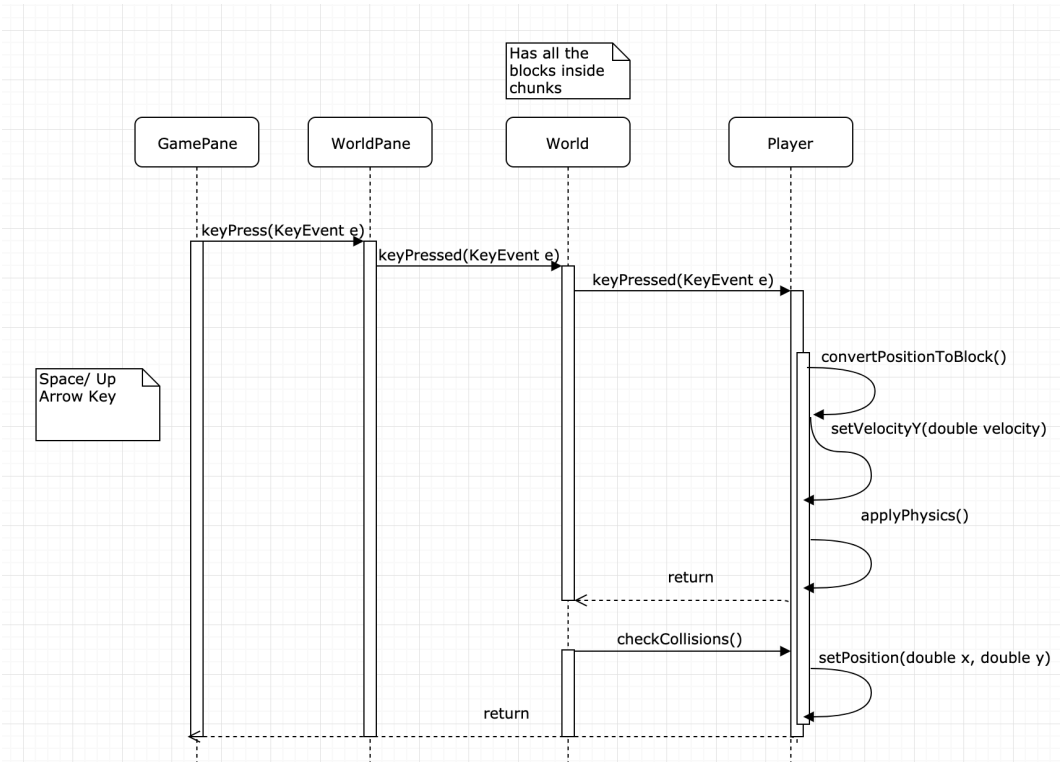
(Differences: can place blocks but not items. Different callback functions)

Class Inventory

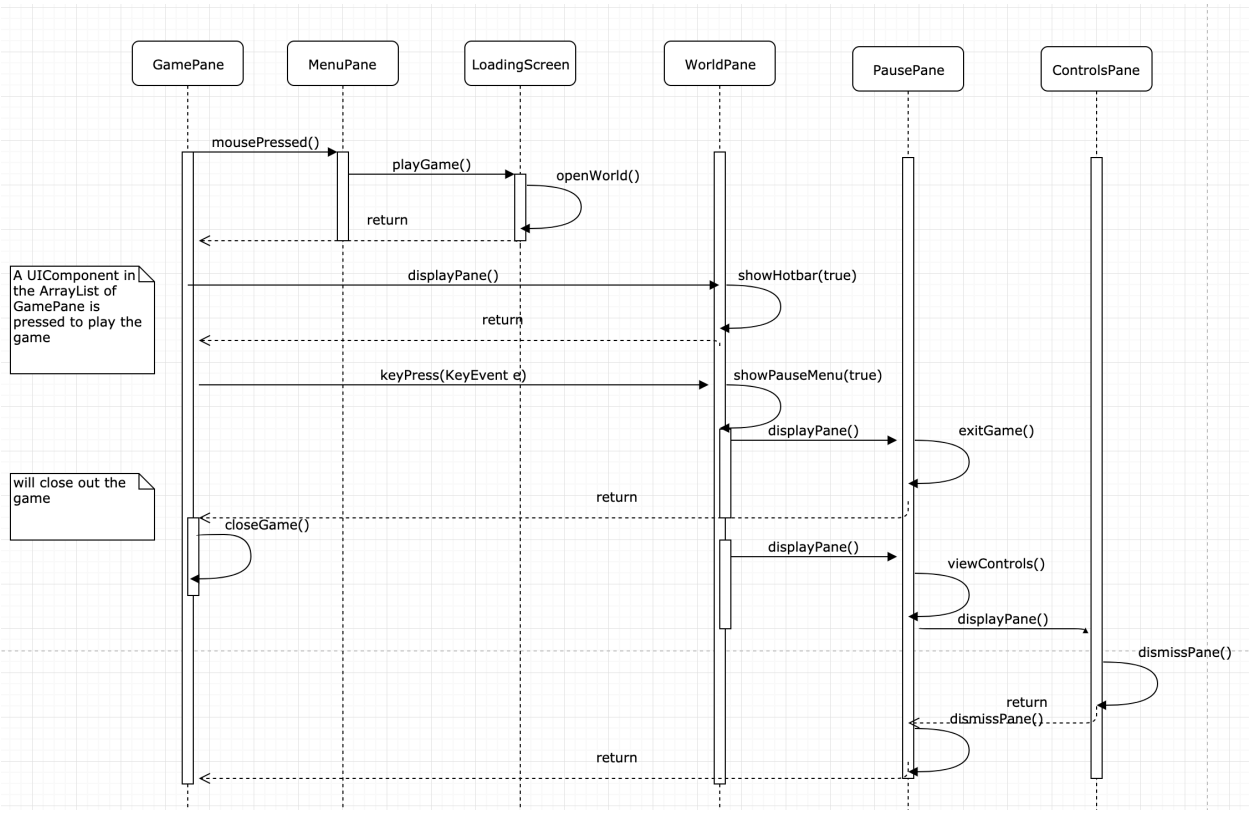
Interaction Diagram

Movement Use Cases

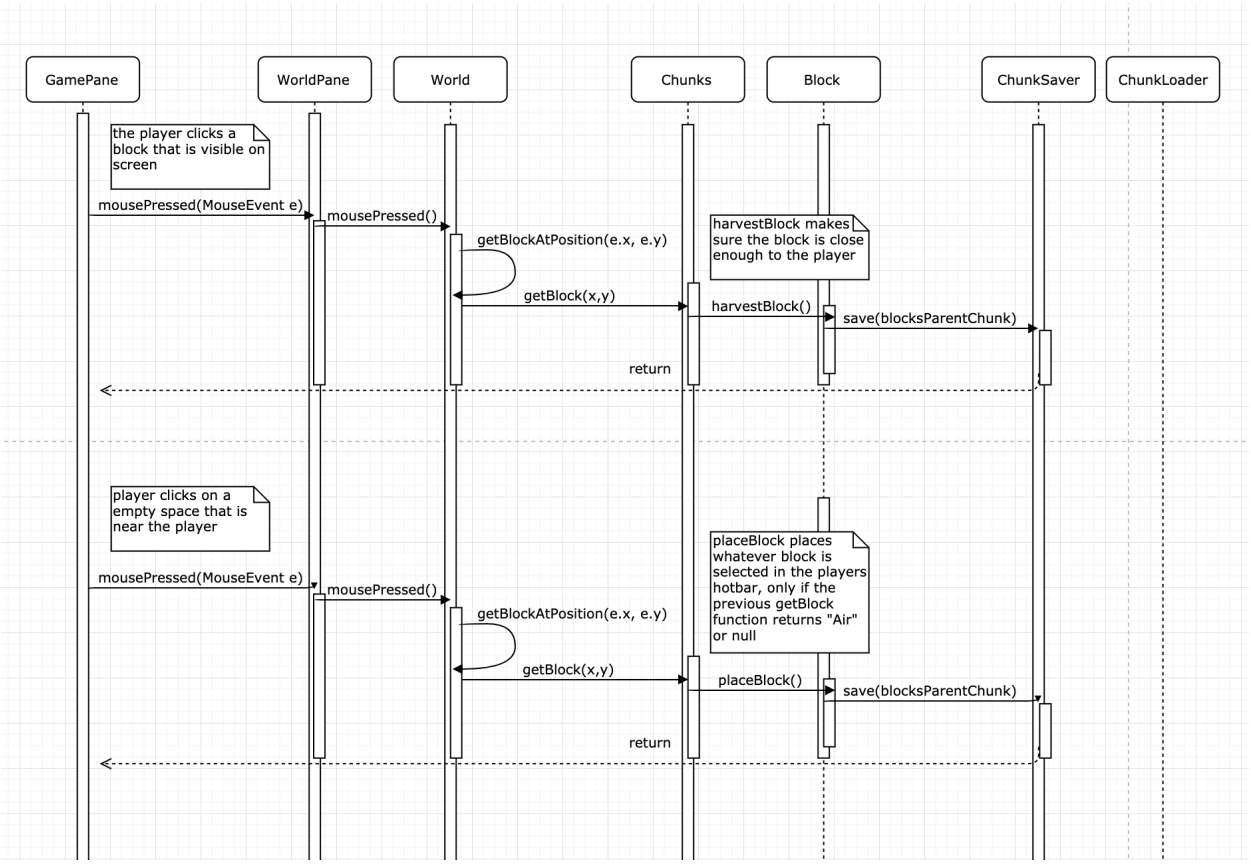




Player UIScreen Navigation Use Cases



World Interaction Use Case



System Architecture and System Design

Mapping Subsystems to Hardware

Our system will not run on multiple computers because server and client processes are able to run one computer.

Network Protocol

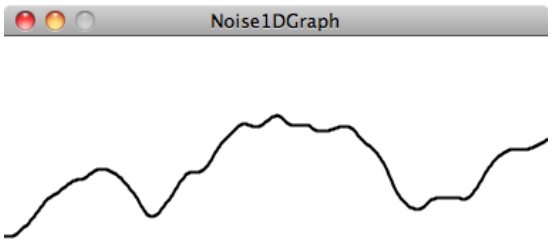
Network protocol does not apply for this project since this program only runs on a single machine.

Hardward Requirements

Our system relies on the computer that its running on to have Java installed. The screen size needed to run this project is 1280 x 720 px for now until we add support to scale the display. The minimum screen size we may end up supporting is 640 x 360 px, but the graphics won't be as sharp as higher resolutions. The program needs to have permission to read and save files since the worlds are saved into text files. Must be run on a windows computer. The games saved data file size grows the more the user plays. It is recommended to have at least 1 gb of space for saving data. The player will need a keyboard and mouse.

Algorithms

The World will be generated using perlin noise. Perlin noise smoothly produces numbers between 0 and 1. This can then be scaled to produce a graph like the one below.



Since this game is blocky, the results of this algorithms will be floored in some way and place blocks along the resulting height map. Another application that this function will be used for is to determine the block type. For example, since the results of the function result in a number between 0 and 1, we can set ranges that result in selected blocks. For example, we can set a rule that determines the blocks to be as follows

Numbers 0 through 0.5 => Dirt

Numbers 0.51 through 1 => Stone

Another layer of noise can be applied if the block result is Stone to create ore blocks

Numbers 0 through .8 => Stone (Stone is very common, so stay as stone)

Numbers .8 through 1 => Iron Ore block

PerlinNoise(double x, double y) : Double results (between 0 and 1)

BlockLookUp(int x, int y) : Block (uses the perlin noise function above and the ranges)

Another algorithm that will probably be used is AABB, which stands for axis-aligned bounding boxes used to detect collision. This will be implemented for our players and blocks as a Collidable object. For each frame that gets rendered, we will use acceleration and velocity variables to calculate the next position of the player. If the player is detected to be moving into a Collidable object, we won't move the player into the block, and instead will move the player to the top, left, right, or bottom side of the block depending on where they are at relative to the block. At this point, the players acceleration and velocity may be set to 0 for those directions.

Data Structures

This project will use HashMaps as the ideal data structure. The first application of HashMaps will be to store Blocks, which will look something like this:

HashMap<String,Block> Chunk

The key for the hashmap will be the blocks global coordinates converted to string, in the format of : x%16,y%16

The second application of Hashmaps is very similar to the first. It will look like:

HashMap<String,Chunk> Chunks

The key for the hashmap will be the chunk identifier which starts at 0,0 and adds 1 to the x and y for every 16 blocks. The goal of this HashMap is to be able to access specific chunks, and then be able to lookup the blocks in the nested Chunk HashMap.

The decision to use hashmaps was made because the data structure needs to be able to grow indefinitely, and hashmaps provide this and are a quick way to retrieve a value based on a key.

User Interface design and Implementation

The initial screen mockups had the right functionality idea, but it may be more confusing for the user if its an overlay, so instead of having the user interface have a lot of overlays, we will have them be full screen seperate screens that we display to the user to make it easier to understand and use. This mainly comes into play with the crafting view, and will be its own separate screen instead of just overlaying it on top of whatever the player is currently doing.

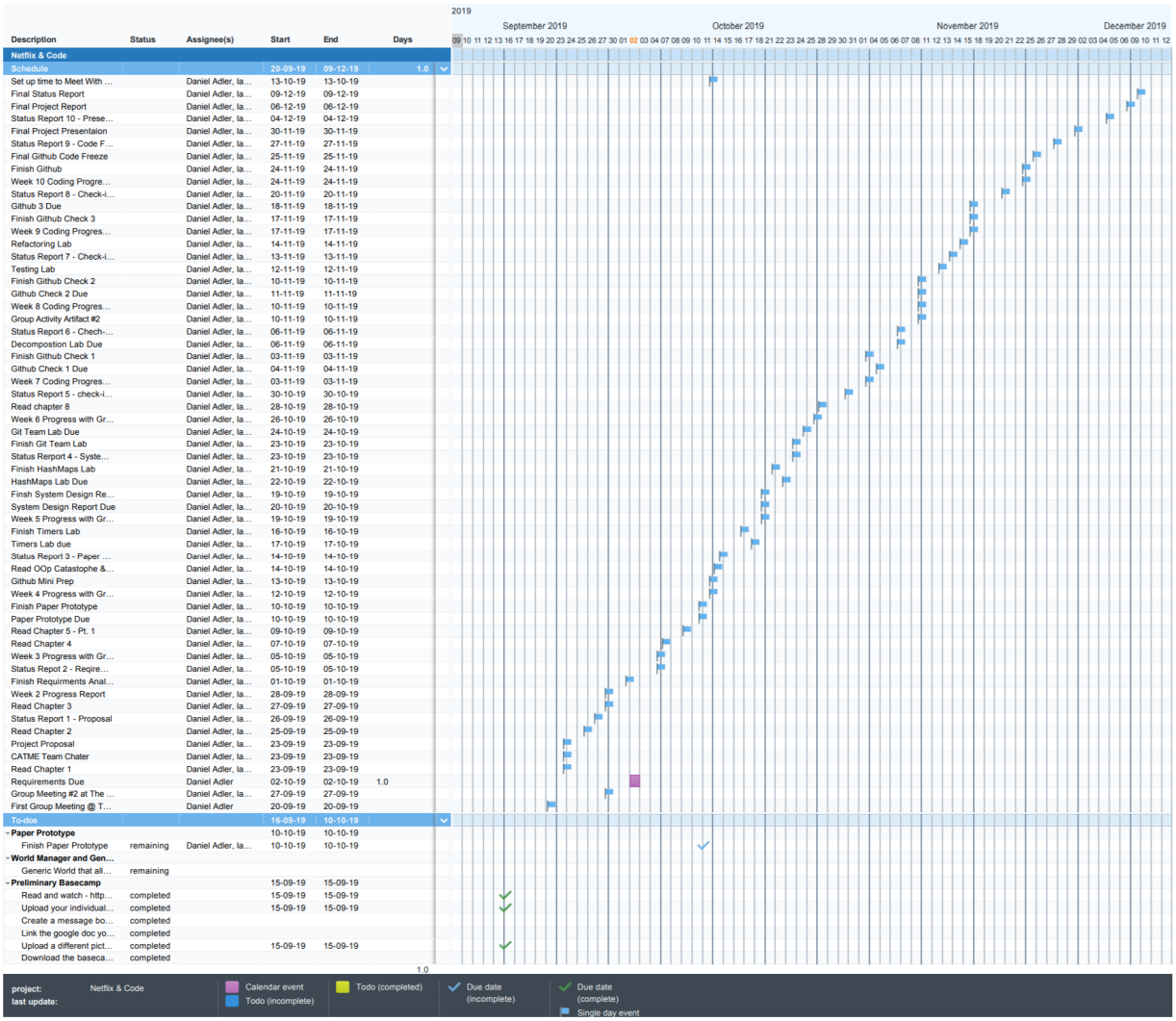
Also, originally, some of the testers for the paper prototype didn't really know what to do. In order to guide the player for the first few minutes of the game, we will be implementing a tutorial box that is overlayed on top of the players view in the GamePane. This hint will tell them to cut down trees, and craft their first tool, and then go away. Also, since the paper prototype was black and white and lacked color, the players were not able to tell what some of the blocks were, like leaves, grass, dirt, stone. They all blended together.

Another thing we will do to ensure they know how to use the build system is have them start on a floating piece of land, and in order to explore the rest of the generated world, they will have to build a bridge to cross the gap.

Progress Report and Plan of Work

Progress Report: So far, we haven't implemented any of our use cases. On the other hand, we have already written the code for world generation and have made the item/block IDs and some of the methods that pertain to the blocks and items. The code for the world generation will have to be changed a little in the future but it should be relatively easy.

Plan of Work:



Ian: WorldScreen, Player, Camera, World, Chunk, ChunkLoader, ChunkSaver, WorldBlock, StorageBlock

James: Game, UIScreen, PauseMenu, UIObject, InventoryPane, UIButton

Daniel: CraftingPane, CraftingUI,CraftingBlock, Recipe, BlockType, ItemType, EntityType, InventoryItem, Inventory

Brandon: ControlsPane, LoadingScreen, Door, MenuScreen

If any other classes need to be created, they will be created based on who works with classes that are the most related to the new class

References

For Terrain generation:

https://paginas.fe.up.pt/~ei12054/presentation/documents/thesis.pdf

https://pdfs.semanticscholar.org/05b2/e79707e961add3d4aceeb12211c2184502c7.pdf

http://benjaminmark.dk/Procedural_3D_Cave_Generation.pdf

These are good sources that provide quality examples on how to generate in an open world game.

Camera Positioning:

https://gamedev.stackexchange.com/questions/46228/implementing-a-camera-viewport-to-a-2d-game

Will help with move the camera, which basically moves all items that are rendered relative to the position of the camera.

Reading and saving to a file:

https://www.quora.com/Whats-the-most-efficient-way-to-design-a-video-game-save-file/answer/Dante-Marshal?ch=10&share=23c02250&srid=aLtDu

Organizing the save file efficiently

https://stackabuse.com/reading-and-writing-files-in-java/

This will be used to save the player inventory and the world that is edited by the player