System Requirements:

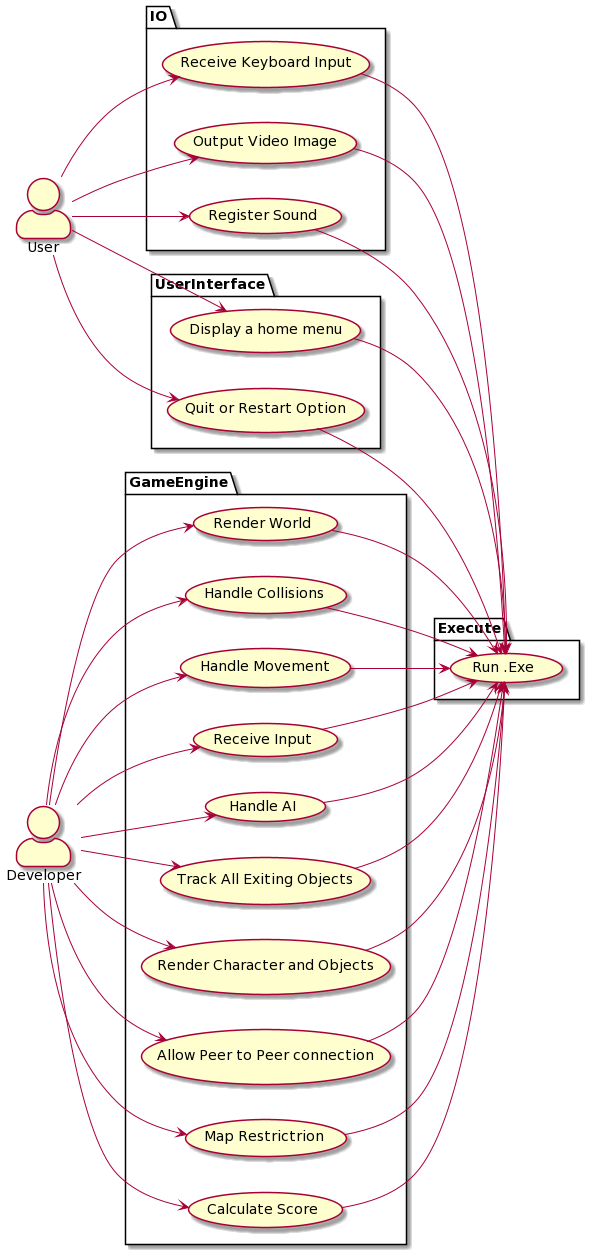
1. Algorithmically generated map for user to play on
   1. Nonfunctional
   2. Developers
2. Generates character and object sprites
   1. Functional
   2. Developers and the Player
3. Should generate sound from the sprites from predetermined files
   1. Nonfunctional
   2. Developers
4. Enables character movement
   1. Functional
   2. Developers
5. Should accept information from peers about where they are on a map
   1. Nonfunctional
   2. Developers
6. System should resolve conflicts between peers about NPCs
   1. Functional
   2. Developers
7. System will recognize collision between sprites and other players
   1. Functional
   2. Developers
8. System will calculate and store scores
   1. Functional
   2. Developers
9. System will broadcast new high scores to all peers through results screen
   1. Nonfunctional
   2. Developer
10. System should draw the environment at an acceptable rate
    1. Functional
    2. Developer
11. System should not allow vessel to leave map by center vessel
    1. Nonfunctional
    2. Developers
    3. Constraint
12. System should Spawn according to rules set in the code
    1. Functional
    2. Developer
13. System should set system where User dies or spawns
    1. Functional
    2. Developer
14. System should determine the angle of impact according to when the User has an collision and calculate damage
    1. Nonfunctional
    2. Developer
15. System should track the health of User and enemies
    1. Functional
    2. Developer
16. System should determine the score by passing the score the UI that directs to the results
    1. Nonfunctional
    2. Developer
17. System shall keep track of the health, location, and score during process
    1. Functional
    2. Developer
18. System should have a reset method
    1. Functional
    2. Developer

User Requirements:

1. User must have access to local network
   1. Functional
   2. Player of the game and peers
2. User must have a method of controlling the playable character
   1. Functional
   2. Player
3. User must have a compatible system to play on
   1. Functional
   2. Player

Use Cases:

1. User Moves
   1. First, the IO system will pick up keyboard input
   2. Second, the IO system will pass the input into some mathematical function that translates the time the key has been held into a speed by which the user should rotate or move forward
   3. The mathematical function (handled in the game engine) will adjust the user’s rotation angle (stored on the user object) or position in the world
   4. The draw function in the Engine (and UI) will retrieve the information from the user object and draw the boat at the specified rotation
2. User calculates scores
   1. The user object will have a running total of the score the user has accumulated.
   2. Each time the user collides with another object (PIW, other vessel, etc), that information will be passed to a function by the engine and the outcome will be added or subtracted to the user’s running score
   3. Overall, each ‘gametick’ the user will pass the score to the UI which will draw the score on the screen
3. Generates World
   1. The game engine will generate a world based on parameters. The world will be generated in ‘chunks’
   2. The algorithm used to generate the world will rely on both set parameters and random noise. The world generation will rely on a seed. Chunks will be handled in the engine and loaded and unloaded as necessary when translating the world.
4. User collides with another object
   1. Handled on the engine level
   2. When a user collides with another object, several factors will need to be accounted for
      1. Angle of impact with respect to the bow of the contacting vessel
      2. Damage calculations? They will be handled with the user’s object and the NPC’s instance.
      3. Score? Handled in the engine and assigned to the user.
   3. Removal of the NPC or player if necessary
5. Player dies or spawns
   1. Creation of a player instance
   2. Location, health, score, etc will all be factors needed to be kept track of
   3. A player will have very little, but the world must have a player
   4. Upon death, a user should be able to respawn. This should reset all player statistics (variables stored on the player)
6. NPC Generation
   1. Must have a location and a type
   2. We must define rules for NPC generation
      1. Generic spawn rules can be defined
         1. Can only spawn in valid locations
         2. Only a certain number of certain entities
      2. Density?
   3. PIW / bonus objects
      1. No health or detriment to the player for collision
      2. Increases score without any negative side effect
      3. Basically will just have a location (parent class) and a sprite (parent class)
   4. Enemies
      1. Must have health (stored in specific instance)
      2. Keep track of angle, speed etc on specific instance



Architecture Description

We first began building our architectural design using the “Pipe and Filter” method. A great advantage of this display is that the flow of data is very easy to follow. A user will begin at one end of the diagram once they start their interaction with our system and then they progress through the “pipes” of our program until they reach the other end with their output. As we finished this model, we found that something was missing. A great pipe and filter architectural model utilizes a very limited selection of controlled inputs in order to achieve a set of very limited outputs. Think of an ATM system. The user will input a card, input a PIN, select withdraw, and then receive their output of solid cash. This does not work as well for a video game program that has many more inputs and outputs of the system. If we kept this model, it is very likely that our game would have run very slowly because it would be processing a large number of inputs in stages in order to implement the desired output.

By switching over to a layered architecture diagram we created a model that is much easier to follow and make corrections as errors arise.