Requirements, Specs, Architecture, and Design Documentation
New Game Team : A Sea Divided

Members:
Karl Bigley-Bodlak(Concept/Database) Devon Steck (Concept/Content)
Rupal Khilari (Client) Toru Nagao (UI)
Anu Aggarwal (Server) Adolfo Von Zastrow (Integration)

Game Summary

- 2-4 player, real-time, gathering/fighting score competition in a 3-5 minute round.
- Players collect the most points on the same map to win the round.
- The player takes the role of an animal in a 2.5d environment. This means the player and environment is modeled in 3D but the player only interacts in 2D.
- Both players begin on the same map as the same animal in their starting zones or “base”. They must gather food and bring it back to their base, in which they cannot be attacked, to score points. Players can only hold a maximum amount of prey before they must return to their base to “score” it
- Players may become an animal higher on the food chain and/or improve their current animal by spending the points they have collected (the same points that determine the win condition)
- One player can attack another player by using an attack. If one player kills the other before the other brings their gathered points to their base, the killed player loses their un-scored points. Players die when their health drops to zero, and respawn after a short delay in their base. Players can only recover health by going back to their base.
- (Priority 2) Players use stamina to give their animal a short burst of speed, use attacks, and a special ability. Stamina refills slowly over time, but can also be refilled by going back to the player’s base. Stamina is determined by the animal’s metabolic rate
- (Priority 2) Each player will have a radar in their camera view, giving basic information about animals or objects outside of their camera view.
- (Priority 2) During the game, a simulation of an undersea ecosystem will be running. As players consume other species of plant and/or animal life, less of these prey will swim into the map until the player goes back to score their points.
- (Priority 2) Various events could happen throughout the game, which affect the environment, (human fishing, whales, predators appear from off screen)

Game Concept (walkthrough of play)

(Note to whoever does this: feel free to change to whatever way could better demonstrate the elements at work throughout the game cycle)

1.Pre-game
- The player is given a picture of their food web with prey and predators of their animal

(P2) -players can choose their starting animal

2.Start game
- (P2)players get a starting sequence (3,2,1, go!)
- each player starts in their base

3.Mid-game
- NPC fishes and plants spawn in based on the prediction model used by World of Balance
- Players collect prey and attack each other to get points and disrupt the other player, while avoiding hazards like predators.

4.end-game
- the server notifies the players who won the game, and gives rewards.

5.post-game
- (P2) results screen

High-Level Requirements (Prioritized)

CLIENT:

Priority 1:

- Allow the game to be launched through a button in the Lobby's game list. We will use the Lobby's Multiplayer game room to look for players waiting to play the game. One player initializes a game and waits for the other player. The first player will have the option of choosing the species ('Mackerel' will be the only option for both players with a fixed food chain in Priority 1).
- Pre-game scene to educate the players of its food chain and the controls, abilities, and objective.
- At the start of the game, read from the server and assign both players a predefined population of species for their ecosystem (Based on the biomass).
- Set the timer for the game and notify the server of the start and end of the round.
- Spawn/remove random fish species (out of the camera view) based on the biomass predictions needed to maintain ecological balance. Recalculate the count depending on the frequency of updates.
- Maintain (and display) a buffer to hold the 'unscored' points each time a player consumes the right prey. The player incurs a penalty if wrong prey are consumed.
- Re-calculate stamina, health and score each time the player returns to the base. Send these updates to the server – so that other players know their opponent's score status.
- Manage collisions and fight interactions between players – use different key bindings for biting, whacking, consuming, toggle camouflaging.
- Allow for in-game upgrade to become a species higher in the food chain by using a part of their game score. Also consider stamina, speed, health upgrades.
- Develop controls simultaneously for the mobile – aim at parallel development for the mobile, which will reduce the overhead of porting it at a later point in time.
- Display the post-game result – declare the winner and loser. Display a brief summary of how the player fared in comparison to the opponent. Save the biomass reward for the player's species which will be used in in-game ranking.
- Communicate the final score to the lobby so that the player could accordingly earn more credits and have their environment score calculated.

Priority 2:

- Allow the user to choose their player from a set of species at the beginning of the game.
- Develop random external events - falling of fishing nets, oil to challenge the player further.
- Attempt to tie in metabolic rate with the player's consuming capacity (thus limiting it for a period of time), swimming speed, stamina to make it more challenging for the player.

SERVER:

Priority 1:

- Set up a separate server for the game (like Running rhino and cards of wild), which requires a jar file at Lobby server to instantiate itself on client request
- Server will accept requests from and send responses to clients
- Server shall support at least two players i.e. allow multiplayer
- At the start of the game, client request server to start game, get opponent's information and ecosystem model. Thus, server communicates with the DAO layer (and ATN machine) to get the required information and send it to the client
- Send/get timed updates to/from the clients about changes in player location, movement, information to spawn/decrease species
- Server shall get updates about the client joining, leaving or winning the game. Thus, shall respond to the updates accordingly
- Server shall destroy thread after the game ends and players decide to quit

Priority 2:

- Connect more than 2 players on a single thread of game (allowing an ecosystem where many players can play and interact)

Functional Requirements

Client high level requirements for Scenes, objects, functions:

1) SelectionScene
    a) setPlayer() - Player selects the main character they will be playing which is persisted throughout the game, unless they upgrade to another character
    b) showGameplay() – Displays the scene containing player information and game instructions.
2) GameplayScene
    a) showGameplay() – Here the Player is given information about its own species, its food chain, controls, its special abilities for combating predators.
    b) loadGame() – loads the main game.
3) GameScene
    a) startGame() - called by the lobby to launch the game once both players are ready.
    b) createGameMap() – generates a random map with a few obstacles along with the player's base.
    c) initGame() – initializes the player's position, its health, stamina and score.
    d) getHeartBeat() – communicate periodically with the server to determine the player uptime, and transfer opponents score updates.
    e) updateStamina() – updates the player's stamina value based on its movement.
    f) endGame() – gracefully handle end of the game when one or both players exit.
    g) showResultsScene() – display the final scene once the timer is up.
    h) showUpgrades() – list the available upgrades that can be bought with the in-game score – animal upgrades, health, speed.
    i) MapObject
        i) set/get methods for attributes including - dimensions, obstacles, density of plant life, density of rocks.
    j) PlayerObject
        i) consume() – Attempt to eat the animal that it is in contact with.
        ii) move() – Move in 2D space based on arrow keys.
        iii) combat1() – perform an action such as bite, handle health/stamina reduction as required on collision.

        iv) combat2() – perform an action such as whack, handle health/stamina reduction as required on collision.
        v) defend() – perform a defence mechanism action.
    k) EnvironmentObject
        i) drawPosition() - position AnimalObjects based on fuzzy logic code. Decision on predation and movement will be controlled by it.
        ii) interpretSimResults() – interpret results of simulation from the server to spawn/remove species based on the results.
    l) AnimalObject:
        i) updateBiomass() – called when the player eats this prey, reduces its biomass from the total biomass. Updates the player score and reward.

4) ResultScene
    a) showResults() – displays the score, winner/loser, explains the final result of the player's ecosystem. Displays the high scores/in-game ranking.

Server PROTOCOLS and FUNCTIONS:

1. InitializeGameRequest – Receive join request from the client, connect two clients on a single thread of game, get the required information like player and biome data

    ConnectPlayer() - Connect players to the server and create game thread

    IntiatiatePlayer() - Create player Object

    GetPlayerInfo() - Get Species, biomass, id related information about the player

    GetEcosystemInfo() - Get Biome data

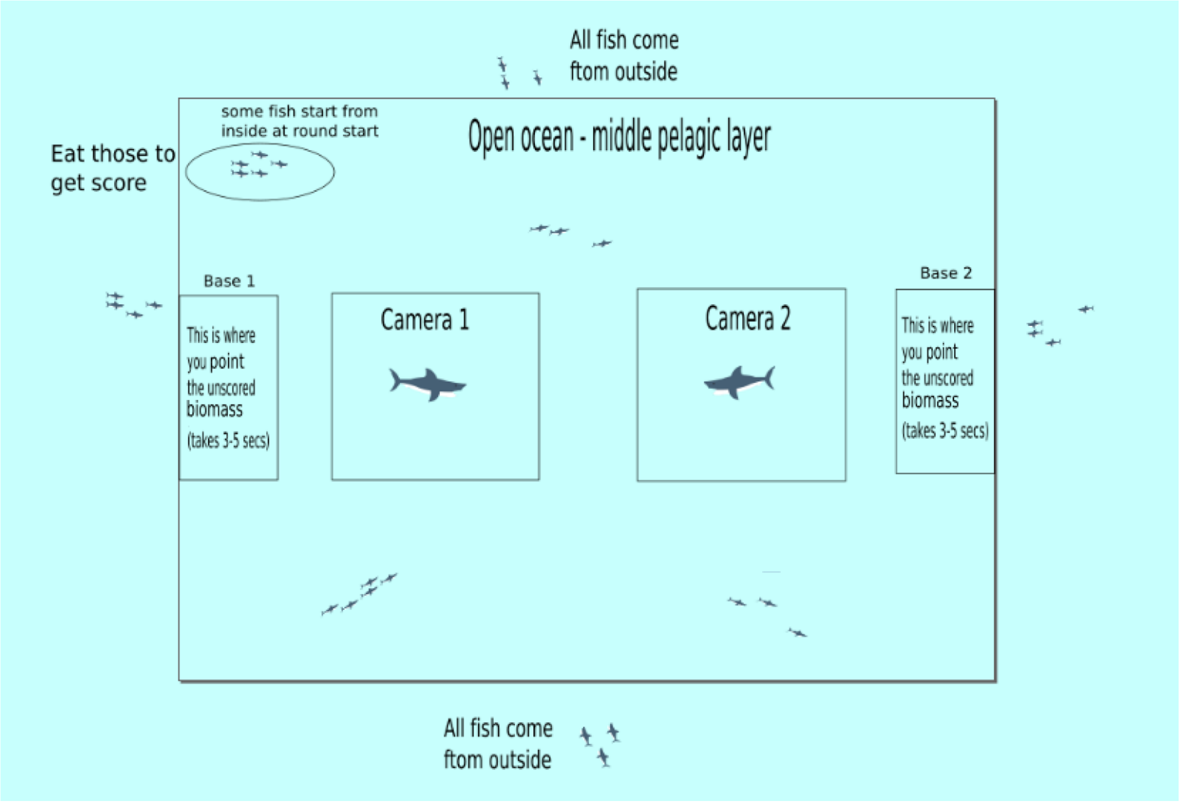
2. StartGameResponse - Send data to the client to set up the game environment
3. UpdateInfoRequest/ Response –
    - Change in the environment and location of the player
        - Request regular updates on the changes in the client side environment
        - Update other player of the changes

        SendHeartBeat() – communicate periodically with the client to send the player uptime, and transfer opponents score updates.

        GetAndSetMovement() - The server routinely receive the information of moving and item collecting from each client, and send to the other.
    - Player Leaving/ joining the game
        - Regular check on player connectivity
        - Send Response to all the clients about change in connectivity of any player (at any point of time)
    - Get Data()/ Save Data() from/to the DAO on client request
4. EndgameResponse– Destroy game thread when game ends [EndGame()]
5. GetAndSaveResult() - After the game ends, get result from the client and save the game result to the database

UI Mockups

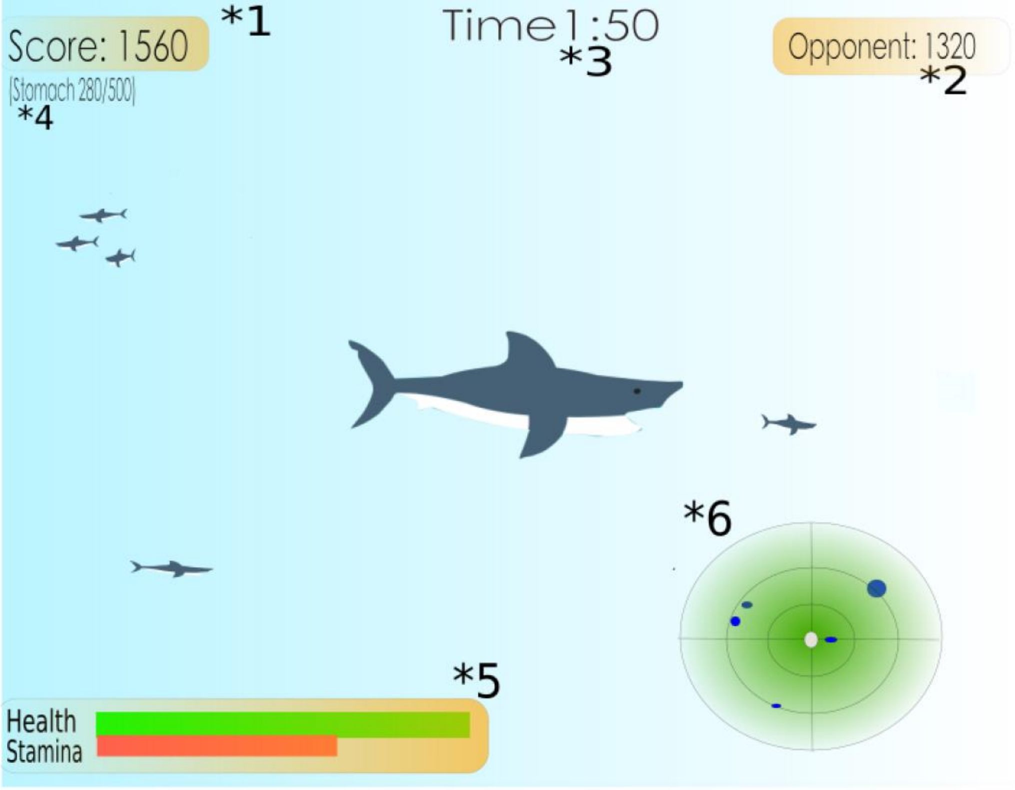
1. General map view


All fish come ftom outside
Open ocean - middle pelagic layer
some fish start from inside at round start
Eat those to get score
Base 1
This is where you point the unscored biomass (takes 3-5 secs)
Camera 1
Camera 2
Base 2
This is where you point the unscored biomass (takes 3-5 secs)
All fish come ftom outside

An overview of the entire game screen.

- Each player has each own camera that tracks and moves with the player.
- Generally, approximately 1/10th of the map with 2 players.
- However, the map could be bigger with more people.

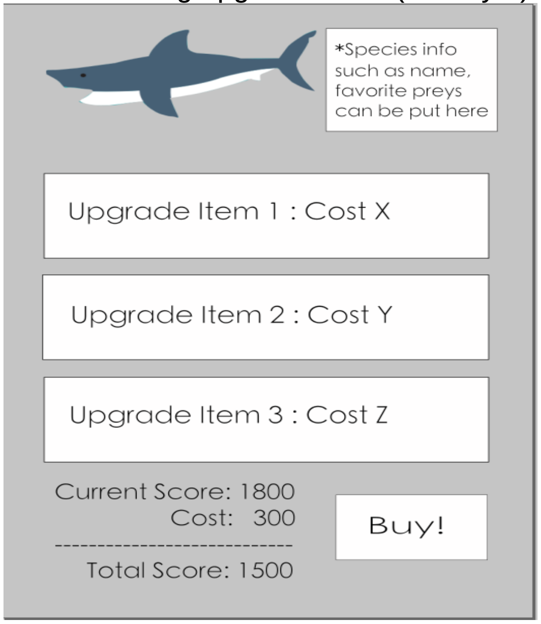
2. Player Camera view



The camera centers the player, and it will contain 6 UI elements.

1. Player’s current score. Underneath of the current score,
2. The opponent’s current score.
3. The remaining time of each round.
4. Stomach parameter which indicates the current biomass that the player is carrying and the max capacity of stomach. In this case, the player’s max stomach capacity is 500 and is currently carrying 280 biomass points.
5. Player’s current health and stamina bars which can be recovered in their base.
6. A mini radar that works as an active sonar of marine species. The radar can detect preys, predators, and the opponent player within a certain radius around the player, and they are displayed as blue dots.

3. Purchasing upgrade items (Priority 2)



In their base, players can also purchase upgrade items using their score in addition to recovering their health and stamina.

- On the top of this screen, the player can see information of the current species.
- The player can choose several upgrade items such as increasing movement speed, health and stamina, and their attack strength.
- The screen shall display the player’s current score, cost of upgrade items, and the player’s total score after purchasing the items.

High Level Architecture and Database Organization

Client - what each player uses during a game, contains more temporary data

Game
- time left in round, other general purposes for setup and closing of game

Player
- holds players' game information: unscored points, max health, max stamina

Map
- positions of players and other animals

AnimalInfo
- Information regarding other animals in the game session, including spawn rate, speed, and metabolic rate.

Server - updates players with information to be shared. Keeps permanent data.

Game

Player

PlayerAnimals
- Info about animals player can become

Map

AnimalInfo

NPCAnimals
- Info about NPC animals spawned in-game

Records
- storing game results

Table Usage: High Level

-The game shall access the table of accounts in World of Balance to retrieve player information and give rewards to the player based on game performance.

-Certain tables, such as Map, Player, and AnimalInfo, are temporary tables used for the duration of a game session between players. These shall be created for each new game session, and removed after the session is concluded.

-the server shall keep information about other fish to be created during a game session in the NPCAnimals table. These fish include prey and predators for the players to interact with.

-PlayerAnimals shall keep information about animals that the user can play as in a game, as well as animals the player can change into using points.

-The server shall update both players with information which it will store in its database temporarily for the duration of a game. The temporary information from a game will be recorded at the end of the game in a table for storing records of matches, which could be used for ranking purposes and/or post-game results.

The new Simulation Model: Animals to be Used

This game will use the World of Balance simulation model in a new, oceanic environment, which World of Balance currently does not have species information to support. We will be making new species to add to the WoB simulation model. A general list of the species to add are listed below:

Species to consider using Pacific environment (Hawaiian)

Near-top predators - (only large sharks or other near-top predators eat these)

- Tuna (upper predator, only eaten by sharks and marlins)
- Lancet Fish
- Marlin
- Smaller shark - Spiny Dogfish (Mud shark)
- Octopus
- Dolphin

Above intermediate, but not quite top

- Lionfish

Intermediate predators

- Squid
- **Mackerel** - The first priority to develop this semester for player use.
- Smaller fish

Filterers / Plankton eaters

- Lanternfish
- Sunfish

Primary Consumer

- Zooplankton

Primary Producers

- Algae
- Seaweed
- Phytoplankton

The Simulation model: Applied Usage

Things we could affect using a Simulation:

- Biomass of animals (decrease biomass of eaten ones, increase ones that eat, according to a threshold of biomass needed to sustain a consuming population.)
- Point values of different animals
- **Spawn rate of animals/plants –** our goal to develop
- (Priority 2) Types of events that occur (feeding frenzy? Bonus for killing other player?)

For our first goal, we want to develop the simulation to affect spawn rates of NPC animals within the game space. After creating the species to use for the World of Balance simulation model, we will run the simulation during a game to help determine spawning of NPC fishes and plants for the players to hunt. The server shall send the simulation request approximately every ten seconds to get data on the biomass of species in the ecosystem, corresponding to a month of change in the simulation. It compares the predictions with the number and type of fish in the game space, and creates more species to be added to the space appropriately.

Schedule for Deliverables

Due dates, start - end
3/15 Development start / presentation given in class
3/19 Concept presentation turned in
3/23 Requirements and specs finalization / research and learning the tools required (ongoing)
3/23 - 4/14 Prototype written: basic premise, 2 players can connect might be playable
4/14 - 4/26 Full game written, ought to be playable
4/26 - 5/3 Integration: bug fixing of game code / adding additional content
5/3 - 5/10 Bug fixing of integration/preparing final presentation and documentation.
5/14 (or final presentation date) development end