Rialen Naidoo and Melubuhle Mduduzi Gula-Ndebele

GADE

Part 2

Table of Contents

[High concept statement 2](#_Toc165624870)

[Features 2](#_Toc165624871)

[Gameplay Loop 2](#_Toc165624872)

[Initializing the Game 2](#_Toc165624873)

[Handling Input 2](#_Toc165624874)

[User Interface 3](#_Toc165624875)

[Rules 4](#_Toc165624876)

[Game Rules 4](#_Toc165624877)

[Objective 4](#_Toc165624878)

[Setup 4](#_Toc165624879)

[Gameplay 4](#_Toc165624880)

[Special Rules 4](#_Toc165624881)

[Game state 5](#_Toc165624882)

[Game State Utility Function 6](#_Toc165624883)

[Mathematical Equation for Utility Calculation: 6](#_Toc165624884)

[Evaluating Game State Utility: 6](#_Toc165624885)

[Examples of Utility Evaluation: 6](#_Toc165624886)

# High concept statement

This will be a turn-based battleship game for two players. Both players will have to try to sink all of their opponent's ships before their ships are sunk.

## Features

* 2 Player
* Turn-Based
* Battleship type game

## Gameplay Loop

1. Player 1 sets up their ships on the board.
2. Player 2 sets up their ships on the board.
3. Player 1 takes a turn.

* Player 1 chooses a square on the board to attack.
* If Player 1 hits one of Player 2's ships, they get to take another turn.
* If Player 1 doesn't hit one of Player 2's ships, the turn ends.

1. Player 2 takes a turn.

* Player 2 chooses a square on the board to attack.
* If Player 2 hits one of Player 1's ships, they get to take another turn.
* If Player 2 doesn't hit one of Player 1's ships, the turn ends.

1. The game continues until one player has sunk all of the other player's ships.

## Initializing the Game

* Create a board for each player.
* Create a list of ships for each player.
* Place the ships on the board randomly or allow the players to choose their ship placements.

## Handling Input

* When a player takes a turn, get the player's input for the square they want to attack.
* Check if the player has already attacked that square.
* If the player has not already attacked that square, determine if the player hit one of the opponent's ships. If they hit a ship, mark the square as a hit and check if they sunk the ship. If they sunk the ship, remove it from the opponent's list of ships.
* If the player does not hit a ship, mark the square as a miss.
* If a player sinks all of the opponent's ships, the game is over.

## User Interface

* Display the game board for each player.
* Display the ships for each player.
* Display the list of sunk ships for each player.
* Get input from the players for their turns.
* Display the results of each turn.
* **Possible UI Options**
* Console
* Graphical User Interface (GUI)

# Rules

## Game Rules

### Objective

The objective of the game is to sink all of your opponent's ships before they sink all of yours.

### Setup

* Each player takes turns placing their ships on their own game board.
* Ships can be placed horizontally or vertically, but not diagonally.
* Ships cannot overlap or touch each other.
* The following ships are available to each player:
* Cosmic Carrier (5 cells)
* Star Battleship (4 cells)
* Nebula Cruiser (3 cells)
* Solar Submarine (3 cells)
* Comet Destroyer (2 cells)

### Gameplay

* Players take turns guessing the location of their opponent's ships by calling out coordinates on the board.
* If a player guesses correctly, they hit a ship. The opponent must mark the hit on their board.
* If a player guesses incorrectly, they miss. The opponent must mark the miss on their board.
* A ship is sunk when all of its cells have been hit.
* The first player to sink all of their opponent's ships wins the game.

### Special Rules

* A player cannot guess the same coordinate twice.
* A player cannot guess a coordinate that is outside of the bounds of the board.
* If a player hits a ship, they can continue to guess coordinates around the hit ship until they miss or sink the ship.
* If a player sinks a ship, they get an extra guess

## Game state

1. **Game Board Representation**: The game board can be represented as a 2D grid where each cell corresponds to a position on the board. Each cell can store information about whether there is a ship present, whether it has been hit, or if it is empty.
2. **Ship Placement**: The positions of the ships on the board can be stored in a data structure such as a list or dictionary. Each ship can have attributes like its type, size, orientation, and health status.
3. **Hits and Misses**: Tracking hits and misses on the board is crucial for determining the outcome of an attack. This information can be stored in a separate data structure to record the coordinates where hits and misses occur.
4. **Game Status**: Game-related information such as the current player's turn, the game outcome (win, lose, or draw), the number of ships remaining for each player, and other status updates can be stored in variables within the game script.
5. **Example**:

* At the start of the game, the game state includes the initial placement of ships for both players.
* After each player's turn, the game state is updated to reflect the results of the move:
* Hit or miss information is recorded.
* If a ship is hit, its health status is updated.
* Game status is checked to determine if a player has won.
* Turn switches to the next player.
* The game state is continuously updated throughout the gameplay, reflecting the evolving state of the game board and relevant game information.

By storing the game state effectively, the Battleship game can accurately track the progress of the game, handle player actions, and determine the outcome based on the game rules and board interactions.

# Game State Utility Function

The utility function needs to consider factors such as ship placements, attacks, board control, and game progression in a 3D environment. Here's an outline of how the utility function could be formulated:

## Mathematical Equation for Utility Calculation:

The utility calculation in a 3D Battleship game could involve a weighted combination of various factors that influence the desirability of a game state. Factors could include:

* **Ship Health**: The health status of ships can be a critical factor influencing the utility of a game state.
* **Hits and Misses**: Tracking successful hits on enemy ships and misses could affect utility.
* **Board Control**: The control of key areas on the game board can be weighted in the utility function.
* **Strategic Positioning**: The positioning of ships and potential attack strategies can also impact utility.

The utility function in a 3D Battleship game may be represented as:

Utility(state) = w1 \* factor1 + w2 \* factor2 + ... + wn \* factorn

Weights w1, w2, ..., wn are assigned to each factor, and factor1, factor2, ..., factorn represent the individual factors contributing to the utility value.

## Evaluating Game State Utility:

To evaluate the utility or value of a given game state in a 3D Battleship game, the AI would:

1. Calculate the utility value based on the factors relevant to the game state.
2. Assess the impact of ship positions, hits, misses, and strategic possibilities in the 3D space.
3. Make decisions based on maximizing the utility value to progress strategically in the game.

### Examples of Utility Evaluation:

1. **Ship Health**: Higher utility for a state where player's ships are healthy and the opponent's ships are damaged.
2. **Hits and Misses**: Positive utility for states with successful hits on opponent ships.
3. **Board Control**: Favourable utility for states where the player controls key positions on the game board.
4. **Strategic Positioning**: Higher utility for states with ships in advantageous positions for attacks and defences.

By calculating the utility value of game states in a 3D Battleship game, the AI can make informed decisions to strategize its attacks, defences, and overall gameplay to increase its chances of winning.

# Link to repo

<https://github.com/melub98/GADE7321-2024-POE-2>