ASTROID INC.

High Concept Document

Dylan James Ramsden

Contents

High Concept

The player/'s finds themselves in 2095, where there has been a vast growth in the competition and drive to mine planets and asteroids for minerals, birthing the first generation of trillionaires. Players take control over the worlds superpowers, USA and USSR, scourging the galaxy for planets and asteroids to mine and harvest in attempts to out sustain and produce higher profits than their competition. Be warned, these superpowers have been found to destroy and overcome anything in their path to success and wealth. Will you be the strongest, greediest, and wealthiest superpower mankind has ever seen?

Rules

Aesthetic:

- The game follows a realistic art style
    - Attempting to aesthetically create an experience that allows the player to feel as though they are the commander of a real space mining fleet
- The game is viewed at a 3D orthographic angle
    - The player is able to use the mouse to rotate the camera around the game board, viewing it from whichever angle they choose
- The lighting of the game is very minimal, using light sources from stars (e.g. the sun) and a default directional light to ensure everything on the board is clearly visible at all times, and from all angles
- Each country's ships and related assets are assigned a colour
    - USA is blue
    - USSR is red

Sound:

- The sound throughout the game follows a very minimalistic theme
    - There are no overpowering sounds
    - The idea is to simulate the silence and ambience there would be in space, with the only sounds being from man-made happenings (e.g. An exploding ship)
- Sounds can be muted at anytime throughout the gameplay and navigation through menus

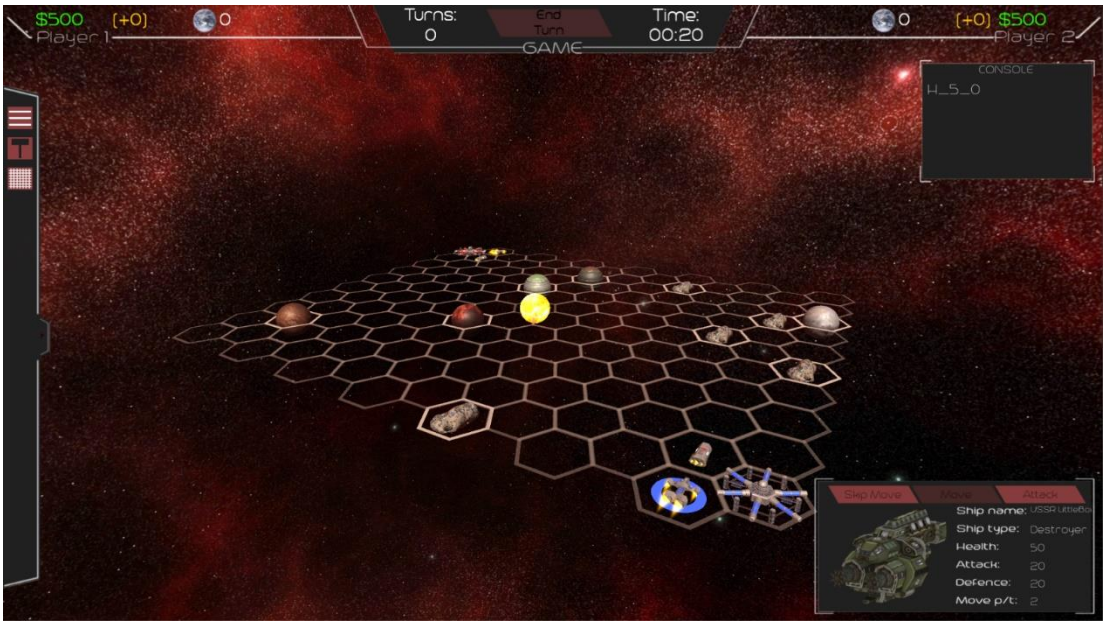
UI (User Interface):

Navigation/Menus:

- The navigation and menus within the game are designed to be as simple and user-friendly as possible
    - Ensures that the user is aware of where they are, in terms of the game's navigation hierarchy, at all times and aware of where each button will take them

Gameplay scene:

- With the gameplay scene having many UI components, holding lots of information as well as providing different functions for the players, it is ensured that they are all grouped based on the function and information they provide
    - This ensures that the player/'s becomes familiar of where to find certain functions and information based on what they need done and are looking for
    - Another important function that the categorized UI provides is preventing the user from being overloaded from information, only providing them with information when needed



(Note. This UI design is a work in progress and may change when the game is released)

- UI groups include:
    - Information panel *(Top of the screen)*
        - This bar stores each players information e.g. their total money, income per a turn, mineables owned and ships owned
        - Also provides information about the current game e.g. the number of turns that have been played and how long the game has been in progress
    - Console (*Top right of the screen, just under the information panel*)
        - The console provides the player with information based on what their mouse is hovering over e.g. if the player is hovering their mouse over a planet, it will provide them with information based on that planet (name of the planet, resources it provides per a turn etc.)
    - Navigation panel (*Left of the screen*)
        - This panel stores and provides the player with many features that they can use within the gameplay (e.g. The shipyard which allows the player/'s to purchase new ships) and with navigation features (a pause menu) to allow the player/'s to pause the game and navigate back to the games menus if chosen.
    - Ship controls console (*Bottom right of the screen*)

        - This console provides the player with information on their currently selected ship
        - Provides actions that can be selected and performed on the selected ship. These actions are based on the ships board surroundings and type e.g. a mining ships mine action can be selected if the ship is on a tile with a mineable housed within it
    - The Shipyard (*Not visible within the given picture, can be accessed by clicking on the second top button in the navigation panel*)
        - Provides the player/'s with the ability to buy a new ship (if they have enough funds), by selecting the type of ship they want and naming it
    - Pause menu (*Not visible within the given picture, can be accessed by clicking on the top button in the navigation panel*)
        - Provides the player/'s with the function of pausing the game, restarting the game, returning to the main menu, and quitting to the desktop.

Gameplay:

General rules:

- The game is a turn-based strategy board game
    - Max 2 players
- Game mode choices allow the player to either play:
    - A hot-seat game (where 2 players play against each other on the same PC)
    - A game against a state machine AI algorithm
    - A game against a neural network AI algorithm
- The goal of the game is to have generated more resources than your opponent by the end of the set number of turns that can be played
    - The max length of the game is 50 turns
    - A player forfeits when all their ships happen to be destroyed and do not have enough income/resources to buy more

Gameplay rules:

- Each player starts with:
    - No money/resources
    - 1 Mining ship
        - Used to establish mines on mineables (planets and asteroids)
    - 1 Destroyer ship
        - Used to destroy enemy player's ships and destroy the enemy HQ (which prevents the purchasing of enemy ships until repaired)
        - If a destroyer destroys an enemy ship, the ship is removed from the enemy fleet
    - An HQ
        - The spawn point for all ally ships
        - Must be functional (not destroyed) to create more ally ships

- A complete player turn consists of:
    - All player owned ships must have all actions used (unless a ships actions have been chosen to be skipped for the turn)
        - Players must use their mining ship/'s which can either be moved or used to establish mines on mineables across the gameboard
        - Players must use their destroyer/'s which can be used to destroy enemy ships or even attack the enemy HQ, and if successful and having destroyed it will prevent the enemy from training more ships for 3 turns. But beware, the blast from the exploding HQ wipes out all enemy ships within a 2-hexagon radius.
    - During this turn, the player is able to purchase new ships and use these new ship's actions immediately
    - Once all a player's ships have been used, the end-turn button will become available allowing the player to end their turn
- Resources/Income per turn:
    - A player's income per a turn is added on to their total resources/money at the start of their turn
    - If a player obtains a mineable, the mineables resources per a turn will only start to be added on in their next turn
    - Once a mineables resources are depleted, the player will no longer receive resources/money from it

Controls

Sound

Click ‘M’ – Turn on/off game sound

Camera

Hold down ‘Mouse Scroll Wheel’ – Rotate camera to whichever angle you want

Actions

Click ‘ESC – Cancel a chosen action, unselect a chosen ship

Click ‘Left Mouse button’ – to select a ship or preform an attack if the attack action for a ship has selected

Click ‘Right Mouse Button’ –Select/define a movement path for a currently selected ship (the move function can only be chosen if a path has been defined)

Game state representation

From the start of development of Asteroid Inc, it was ensured that the architecture of the game and all its underlying data was to follow an object-orientated design. This ensures that the game can be abstracted from its visual representation and only be stored and worked with as classes, with most classes being structured the same due to inheritance. This makes it extremely easy to store a game state and provide an AI algorithm (in this case, a state machine) with data that can be easily understood and worked with.

Game state class

GAMESTATE
Map *Hexagon[,]*
p1Base *PlayerBase*
p2Base *PlayerBase*
efValue *float*
turnCounter *int*
GameState(int)
EvaluationFunction(int)

The game state is designed to only store essential properties that are vital in ensuring the state machine algorithm is able to fairly evaluate and determine its current position, at all times, within the game. All game states within a given game will be stored within an array and written to a JSON file where the data can be accessed and used, at a later stage in time, for the training of a neural network algorithm and other algorithms if chosen. With the default length of a game being 50 turns, each game will have 50 game states that are stored and represent each unique turns made by the player/'s or AI.

The game state stores:

- A 2D array of Hexagon classes which make up the map
    - Each Hexagon class stores:
        - Their current position in relation to the entirety of the games map
        - Mineables that are found within it
        - A list of all surrounding hexagons
- A PlayerBase class for both player 1 and 2
    - A PlayerBase class stores all information relating to a specific player, this information includes:
        - A playerID defining whether the class refers to player 1 or 2
        - The total money/resources a player has
        - Stores the class of the players HQ
        - Stores a list of Ship_Base classes holding all of the players ships currently in their fleet
            - A ship_Base class holds a ships:
                - X and Y position within the hexagon map
                - The ships HP and Damage
                - The ship type

        - Stores a list of Mineable classes holding all the mineables currently owned by the player
            - Used to get the players ResourcesPerTurn
- A float holding the current evaluation function result of a given player, in regard to the current board state (will be explained in more context in the next section)
- An integer holding the turn number that this game state refers to
- A constructor for the game state
    - This is where the turn number will be passed through
    - The evaluation function will also be run in the constructor to ensure it is all done when the board state is created and initialized

In conclusion, all relevant information needed about each game state is stored in a way where there are no visuals but rather just data that can be accessed quickly and efficiently. This ensures that there is not too much stored, making it easier when analysing and using the data by chosen AI algorithms.

Game state utility function (Evaluation Function)

((CurrentPlayer.TotalResources/3000) * 0.05 + ((CurrentPlayer.ResourcesPerTurn*TurnsLeft)/3000) * 0.05)

–

((OpossingPlayer.TotalResources/3000) * 0.05 + ((OpossingPlayer.ResourcesPerTurn*TurnsLeft)/3000) * 0.05)

=

Value between -1 and 1

Overview of EF

Based on the nature of the game, and the games winning condition, it is vital to base the evaluation function on resources. With resources driving the gameplay and player decisions, knowing where the player/AI is in terms of their resources apposed to their opponents plays a key factor in how the player/AI should behave and what actions should be carried out by them.

Every resource value is divided by 3000 due to the max resource count being 30000. This ensures that all resources can be used in a way where they can be represented as a value over 10 (making it easier to obtain a result between -1 and 1).

EF breakdown

Part 1

1. The current player's, whose turn the game state refers to, total resources/money is divided by 3000, making it a value over 10.
2. The result of this is then multiplied by 0.05 due to it holding half the influence over the current players resource value

Part 2

1. The current players resources per a turn is multiplied by the number of turns left within the game
2. The result of this is then divided by 3000, making it a value over 10
3. The result of the division is then multiplied by 0.05 due to this holding the other half of influence over the current players resource value

Part 3

1. The current player's opponent's resources/money is divided by 3000, making it a value over 10
2. The result of this is then multiplied by 0.05 due to it holding half the influence over the opponent's resource value

Part 4

1. The current player's opponent's resources per a turn is multiplied by the number of turns left within the game
2. The result of this is then divided by 3000, making it a value over 10

3. The result of this is then multiplied by 0.05 due to this holding the other half of influence over the opponent's resource value

Once all individual parts have been calculated:

The result of Part 3 + Part 4 is minused from the result of Part 1 + Part 2, returning the evaluation function result for the current game state.

The Result

The result for the current game state is returned as a value between -1 and 1. The way in which this value is interpreted, is as follows:

- A result between -1 and 0 represents a bad position within the game state. This is due to either having fewer total resources than the opponent, having a smaller quantity of resources per turn than the opponent or both.
    - Values closer to 0 represent mildly bad positions within a game state
        - The state machine will play mildly aggressive, meaning it will prioritize the development of mining ships to take over more planets and will target its military ships at destroying nearby enemy miners.
    - Values closer to -1 represent the worst positions a player can have within a game state
        - The state machine will play overly aggressive, meaning it will prioritize the development of mining ships to take over more planets and will target its military ships at seeking out and destroying enemy miners across the entirety of the map. Military ships also have a chance to seek out the enemy HQ, slowing down their ship production for a few turns.
- A result between 0 and 1 represents a good position within the game state. This is due to either having more total resources than the opponent, a larger quantity of resources per a turn than the opponent or both.
    - Values closer to 0 represent mildly good positions within the game state
        - The state machine will play overly defensive, meaning it will prioritize the development of destroyers to project all currently owned mineables and prevent enemy miners from taking them over. Ally miners will still seek out new mineables to increase the lead.
    - Values closer to 1 represent the best positions a player can have within a game state
        - The state machine will play mildly defensive, meaning it will not prioritize the development of any ships but rather focus on saving money to ensure its end game victory. If there is an increase in enemy ship production, the state machine has a chance to counter this and produce more destroyers to project currently owned mineables.