

Software Engineering Project: Assessment 2

Gandhi-Inc. - Project Blind Eye

Architecture

Now that the actual implementation of the design has begun, we have expanded upon our previous abstract architecture to create a more concrete version, as detailed in this document.

1. Architectural Tools

The software used to create the diagrams has been reviewed again during this phase and we are now using Eclipse Papyrus [1] to do so. The concrete model is more easily produced under the stricter formatting rules than the relatively freeform abstract model [2]. Papyrus is free software allowing a decent degree of malleability, making it suitable for our needs.

As with the abstract architecture, we have used the commonly-used system design model UML to describe our concrete architecture. We have again used a class diagram to display the overall structure of the program and the relationships between the different components. However, the concrete diagram goes into greater detail, incorporating details such as data types and visibility levels.

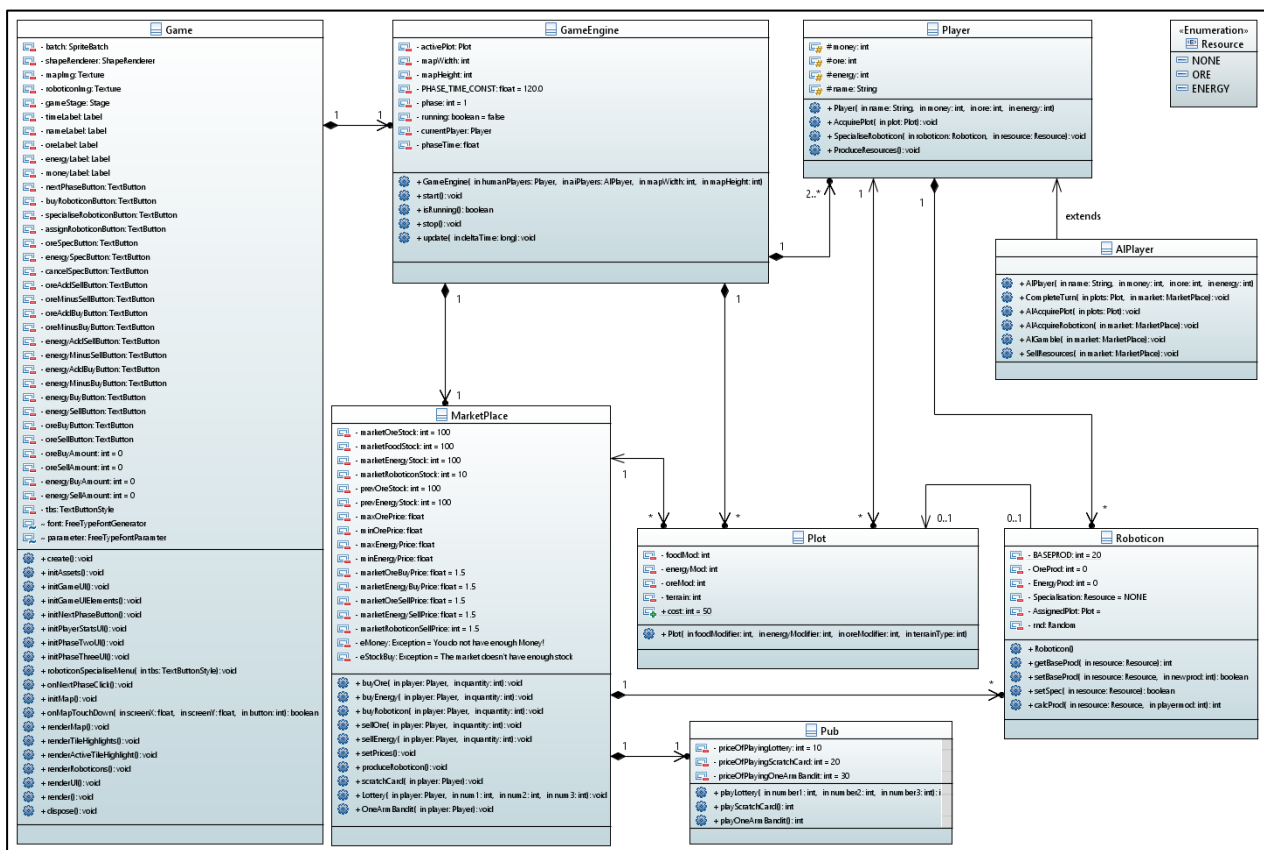


Figure 1 [3]: Architectural Class Diagram.

Note: Most getter and setter methods have been omitted from this diagram to reduce unneeded clutter and improve clarity.

[1] Eclipse. "Papyrus," eclipse.org. [Online]. Available: <https://eclipse.org/papyrus/>. [Accessed: Jan. 23, 2017].

[2] W. Wood, et al. (2016, Nov. 9). "Assessment 1". Downloads – Gandhi Inc. – SEPR Project group [Online]. Available: <http://gandhi-inc.me/downloads/Gandhi-Inc1.zip>. [Accessed: Jan. 23, 2017].

[3] W. Wood, et al. (2016, Dec. 18). "ArchitectureClassDiagram.png". Gandhi Inc. – SEPR Project group [Online]. Available: <http://gandhi-inc.me/downloads/ArchitectureClassDiagram.png>. [Accessed: Jan. 23, 2017].

Software Engineering Project: Assessment 2

Gandhi-Inc. - Project Blind Eye

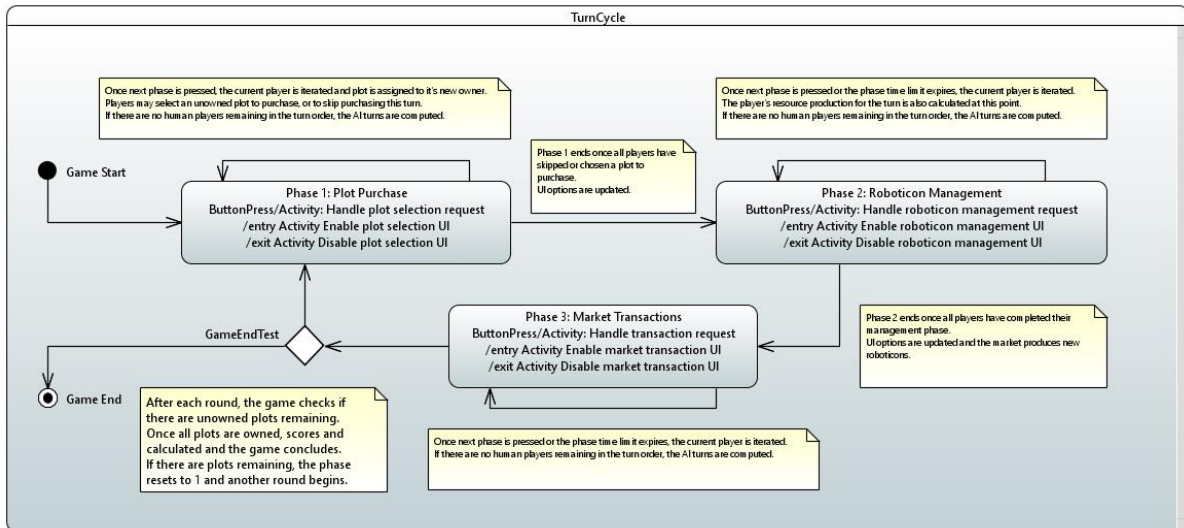


Figure 2 [4]: State Machine.

[4] W. Wood, et al. (2016, Dec. 18). "StateMachine.png". Gandhi Inc. – SEPR Project group [Online]. Available: <http://gandhi-inc.me/downloads/StateMachine.png>. [Accessed: Jan. 23, 2017].

Software Engineering Project: Assessment 2

Gandhi-Inc. - Project Blind Eye

2. Concrete Architecture

2.1. Class Diagram

The concrete architecture builds upon the previous abstract model, incorporating more detail and changing to reflect the actual code now that the first stages of implementation have begun.

Operators and properties now display their visibility level and the data types of their parameters and returned values. As the program has now been implemented in code, the concrete diagram uses the same names and terminology as in the code, where those are different or absent in the abstract model. The relationships between classes have also been expanded upon, displaying their multiplicity and aggregation levels.

Some classes have also been expanded upon or split into more specialised units. The game class from the abstract architecture has been split into two classes, one handling the user interface and graphical rendering, the other handling the actual game logic and data. This modularisation was done to separate the specific graphical implementation from the actual game functionality, allowing it to be more easily changed without requiring substantial modifications to the actual game code.

An extension for the player class has also been added to implement the additional functionality required for AI players.

While not shown in the abstract architecture, all attributes and methods within a class have levels of visibility, determining which classes are allowed to access them. As per programming standard, most attributes are 'private', only accessible within their own class. They are instead accessed through methods provided by the class, allowing for proper validation and management. There are a few exceptions where needed, such as in the player class.

The Game class forms the top layer of the program and handles the user interface and graphics. It contains the graphical elements and is responsible for rendering the various elements of the display and interface. It also initiates and contains the GameEngine class, to which it passes interpreted user input. This allows for the player to actually interact with and play the game.

The GameEngine class contains various game objects, such as players, plots and the marketplace and is responsible for managing the turn phases and active player, as well as calling several automated processes during the turns. It forms the core of the game logic and supplies much of the game's actual functionality.

The Player class stores data relevant to specific players and provides many of the functions for them to actually play the game. Unlike most of the other classes, the attributes are given a protected level of visibility rather than private, in order to allow them to be accessed by the AIPlayer class. The AIPlayer extends upon the Player class, providing the AI logic for simulating a player. The GameEngine maintains one Player for each user playing the game, with a minimum of two required. The Marketplace class provides the functionality for buying and selling resources. It maintains sets of values to determine prices and available quantities of each resource and handles transaction requests by the user. It also contains Pub class, which handles the pricing and functionality for the 'gambling minigames' that are called through the marketplace. Only a single Marketplace exists per game.

The Plot and Roboticon classes have a largely passive role in the program's functionality. They each store a set of values or modifiers to determine their production rate and their methods serve to provide those values when the player calculates their resource production each turn. Plots are all

Software Engineering Project: Assessment 2

Gandhi-Inc. - Project Blind Eye

contained by the GameEngine, but are also associated with their owner and the assigned roboticon, if one exists. Similarly, the Roboticons are contained by their owner, but also have an association with their assigned Plot, if one exists. The game can theoretically support any number of Plots and Roboticons; while Plots and Roboticons must have a single Marketplace or Player as their owner and may be associated with up to one Roboticon or Plot, respectively.

2.2. State Machine

While the abstract architecture was formed primarily of a five-state loop, it has been reduced to three in the concrete architecture and actual implementation. The two roboticon management phases have been merged, while the production calculations and round setup do not require dedicated phases. These functions are instead handled as part of the transitions between phases as they do not require user interaction and do not need to be maintained over an extended time period.

Additionally, the management of multiple players has been changed. In the abstract architecture, the game would run through an entire turn for one player, before moving on to the next. Instead it now iterates through each player in the turn order for each phase before advancing to the next. This allows some operations to be carried out at more intuitive timings within a round without disrupting the balancing between players. This can also allow players more ability to react to each other's actions.

These changes also greatly simplify the turn logic without impacting the functionality of the game from the user's perspective.

The first phase allows players to view and select tiles on the map, providing visual indicators as appropriate. Once a player has chosen an unowned plot to purchase or opted to skip for this turn, the next turn button will confirm and apply their decision, passing control to the next player. Once all players have decided, the next phase begins.

The second phase is for roboticon management, allowing players to buy, specialise and deploy roboticons in order to manage their production of resources. As before, this will iterate through each player before progressing to the next phase. The turn's production for each player will also be calculated upon their completion of this phase

Upon transitioning from the second phase to the third, the market will carry out some functions, constructing new roboticons and ensuring everything is ready for trading to begin. This phase allows players to carry out transactions with the market, buying or selling resources as they choose. Again, all players take a turn before the phase advances.

Once the third phase is complete, the game will run a test to check if any plots remain unowned. If so, it will cycle back to the first phase and the next round will begin. If not, the endgame procedures will be carried out, calculating scores and concluding the game.