Battalions & Besiegement

12/2/2020

Blocker Griffin

Bryant Terry

Guess Crow

Scott Clarke

Thomas Lemmons

Description

Battalions & Besiegement, henceforth referred to as Battalions, is a two player strategy rpg game in which players take turn moving units and taking actions with those units until only one player's units remain.

Management

Blocker Griffin

Bryant Terry

> Undergraduate majoring in Computer Science and minoring in Mathematics. Currently a junior.

Guess Crow

> Undergraduate working towards a BS in Computer Science as well as a minor in Game Design and Development.

Scott Clarke [Team Leader]

> Major: junior in computer engineering. Minor: game design and development. Worked full time as a computer engineering intern at Parsons in summer 2019 and 2020, mainly writing C# for WPF GUIs and interfacing with hardware peripherals following the MVVM pattern. Prior experience with Java includes two years as head programmer on a FIRST Robotics Competition team in high school.

Thomas Lemmons

Work was done online, with discussion and organization being freely had via Discord. The project was hosted and managed via GitHub, with each member pulling the project, working on changes or additions, and uploading them to the master branch. While we did keep a spreadsheet of work hours put into the project, it wasn't rigidly kept up to date. Therefor, we don't know the

exact number of hours that went into the project, but our rough lower estimate is around 65 hours put in between all of us. This is only a portion of the rather generous 225 hours we originally calculated.

Initial Design Effort

8 CRC cards were initially created for this project's design, along with a general use case for the game. Each CRC card was split into responsibilities and collaborators, with each card header representing an overarching class of our project, which were divided in accordance with MVC design. Cards also had an accompanying walkthrough in both general english and in terms of the CRC cards. An example of such design is our CRC card for the Game class. This class's responsibilities were to keep track of turns, stat turns, handle saving/loading of game progress, and determine if a player had won on a given turn. To this end, it collaborated with nearly every other element of the design, such as the Map, the Tile(s), the Player, and the Unit(s)

Design

The project was broadly packaged in accordance with MVC, with a focus on splitting the program into Model, View, and Controller packages.

Data Package

ActionType: Defines the type of action performed by a unit

Location: Defines an x, y coordinate on the map's grid

Orientation: Specifies the direction an object faces

TileEffectFlags: Flags representing what effect a tile has on a unit standing upon it

TileType: Contains information on the type of a tile and how that relates to its effects flag(s)

UnitType: Contains information on the type of a unit and its relevant stats, such as archer, healer, or knight

Models Package

Game: Controls the general flow of the game state, such as players and win conditions

IMapItem: A model for a coordinate on a map

IPlayerItem: A model for an object that is owned by a specific player

ITurnBased: A model for an object that has behavior relating to beginning and ending each turn

Map: Contains a 2D array of tile objects and methods for manipulating the array

MapSelector: Handles selection of objects within the map such as tiles and units

Player: A model for an object that represents the player within the game's logic

SaveSystem: Handles saving and loading the game's state to/from a file

Tile: A model for a tile object, each tile being a space on the map with relevant properties

Unit: A model for a unit object, units being the 'pieces' the player's control during the game

Properties Package

IPropertyChangeNotifier: Notifies listeners when property values of a specific object are changed

PropertyChangeNotifier: An adapter class that implements basic property change notifications.

Util Package

LocationSets: Utility class for performing set operations involving location masks.

Rng: A class that provides methods for random number generation

Stats: A class that is used to calculate slight random variation in a unit's stats/performance

Views Package

GameView: A class that handles the game's overall GUI design

MapSelectorView: A class that handles the user's interactions with the view via their mouse cursor, such as highlighting units and areas when moused over or selected

Sprites: A class that handles the drawing of sprites for tiles and units to the view

Controllers Package

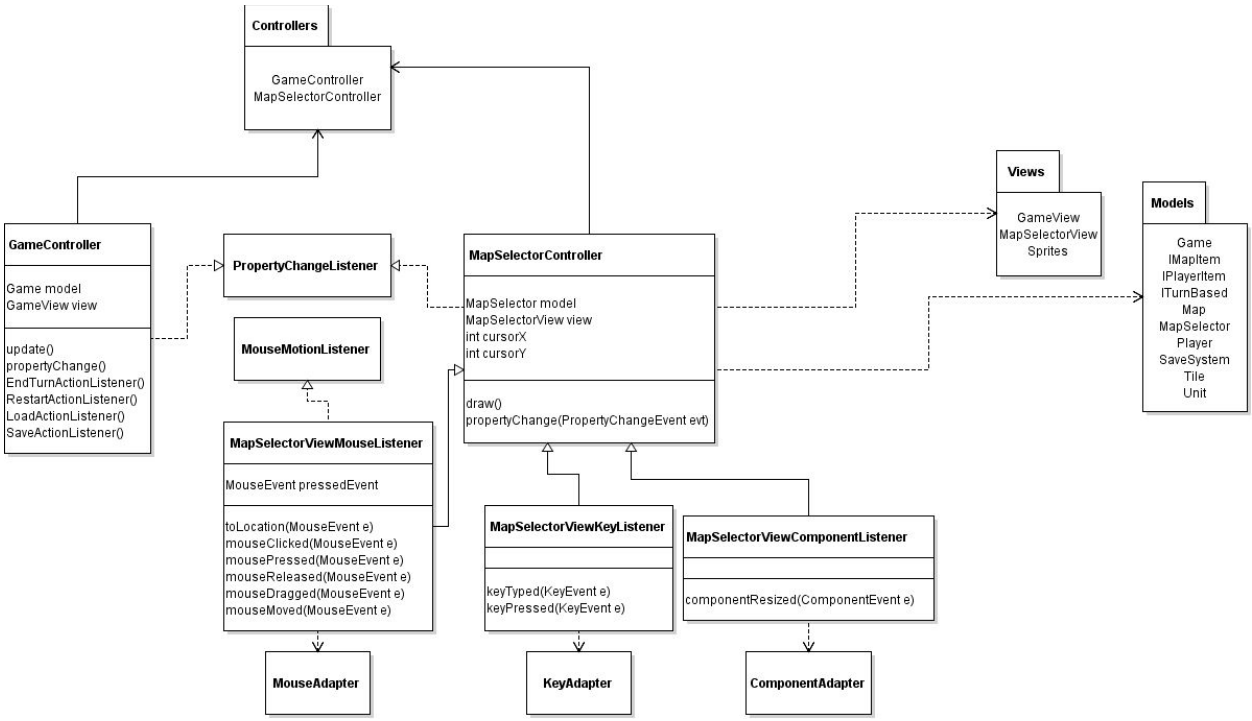
GameController: Controls the flow of the game's state, regulating how interactions from the view affect the model

MapSelectorController: A class explicitly for controlling how the game handle's interactions between the cursor/selection of the view and the game's logic

Image Packages

The image packages consist entirely of image files for the graphic display of units and tiles. Sprites are modified from the work of Andre “Toen” Coppola. His work is distributed for free use under creative commons. You can find his work and information here: https://toen.itch.io/

The following is an example of how the program was organized in UML, focusing on the Controllers package



Implementation

The program is designed and implemented largely with respect to MVC, with components being separated into Model, View, and Controller packages. The Model holds classes that model the game’s objects and data structures, the View handles the display of the GUI as well as the user’s interactions with it, and the Controller handles the interactions between the View and the Model to control the game’s logical flow and procedure. Some packages were not organized into the

MVC framework within the code, such as util, data, properties, etc. but are functionally designed with respect to MVC, and could be further organized with minimal changes to the code.

Testing and Assessment

Testing was done as code was written, as well as each new change was tested after being submitted to the master branch of the project GitHub page. As for the assessment, we believe our project by and large lives up to its original conception. The initial use case was, while modified, broadly followed in the end product. We originally envisioned players playing against a CPU opponent, yet were unable to program such CPU behavior due to a lack of time. General improvements to the program would be fulfillment of our initial secondary goals, such as a title screen, multiple maps, and an overall GUI overhaul with custom sprites and UI elements. That said, we are happy with the work we have done, and satisfied that we've completed a project and a game of this scale (for the first time for many of us).

Appendix

General Use Case

1. User runs the program, and it loads up the GUI with map and both friendly and enemy units

2. The game begins on player 1's turn, the player will select a unit. Hovering the cursor a unit highlights it and shows its stats such as health, strength, defense, etc. on the GUI. Clicking a unit will select it, and selected units show a highlighted range of possible movement tiles. The player may select one of these tiles to move the unit to this spot, or select and enemy unit. Enemies within attack range of the player's selected unit will be highlighted.

3. Player 1 may continue to move and/or attack with their units until they've exhausted all their units. Players may end their turn prematurely by selecting "End Turn". Control of the game will be passed to player 2's turn, in which they will do the same as player 1, and will choose unit(s) to move and/or take an action.

4. The game will continue like this, with the players alternating turns and moving their units about the map, until one player's units have all been defeated. At which point the remaining player will be crowned winner, and the game will end.

Link to CRC Cards, use case, and CRC walkthrough