New Changes in Milestone 2

Created by: Rahul Anilkumar, Christopher Wang, Christophe Tran, Thomas Leung

1. GameState

The changes done to the GameState were to adopt the previous game state for the MVC design pattern. This includes adding listeners to listen for mutations, generating events to transfer information to listeners and updating certain methods to better support a visual rather than a console game.

2. GameStateListener

A GameStateListener interface was developed to allow for the view to automatically re-render when the GameState mutates. In essence, it is how we decided to implement communication between the model and the view.

3. Event

Events are how we pass information between the model and the view. There are three types of events:

a. A PointEvent occurs when a numeric value of the GameState is mutated such as sun points or turn.

b. An EntityEvent occurs when the entity list of the GameState is mutated, this includes when an entity is added or removed.

c. A GameEvent occurs only when the game is over and transfers information of the outcome (i.e. win or lose) to the view.

4. View

The View class is created as a visual element to the game. It uses several arrays of buttons as the primary method of user input. Furthermore, a Command enum is used to store the command that a user selects. Also, the view stores an EntityType enum when a user selects a plant. This is relevant because it means our View contains state.

5. Control

Within the control package, there exists several ActionListeners that handle user inputs and a main class *Control* that attaches ActionListeners to the View and executes the game. CommandListener listens for user input to the command buttons and set the state of the View to a predetermined command type when it is selected. PlantListener listens for user input to the plant buttons and set the state of the View to a predetermined plant type when it is selected. GridListeners listen for user input to a button on the board and performs various methods on the game (model) depending on the state of the view (previous user input).

6. Test

JUnit tests are implemented to ensure that refactors of the model do not break the overall functionality that is expected from the project.