Reflection – Hobby Detectives GUI

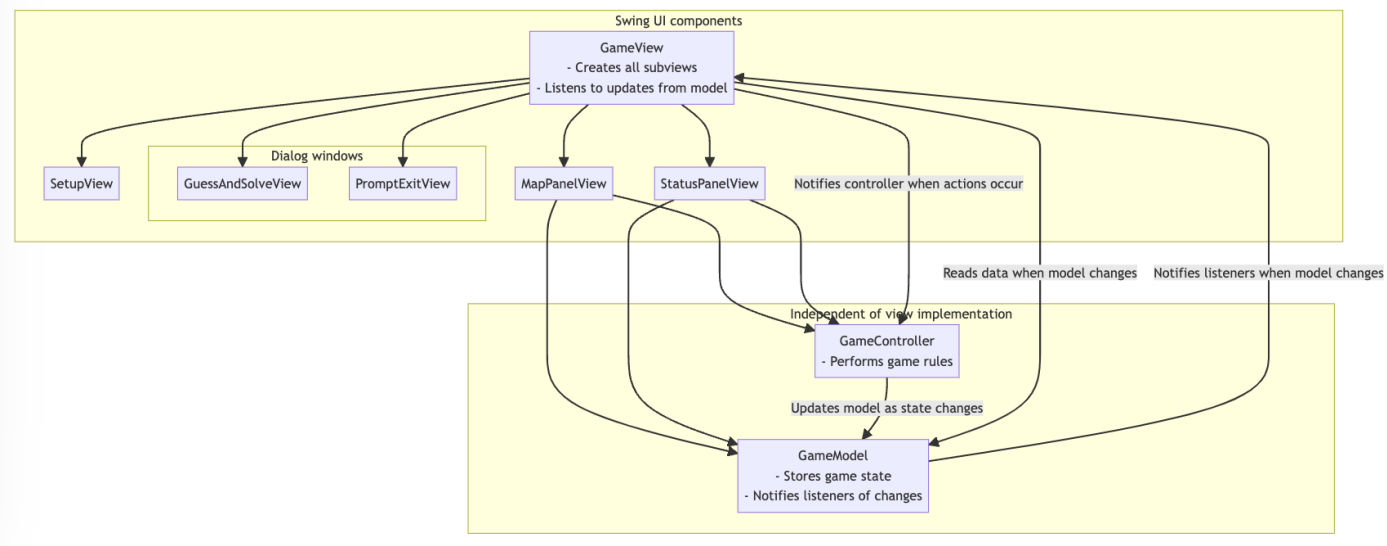
Jackson Rakena (300609159)

Interface

- We used the following components:
  - JFrame – the entire game is inside a master “GameView” which extends JFrame. The game view sets up the GridBagLayout (which we use to layout the left panel and the map panel)
  - JPanel – the left panel (StatusPanelView) and map view (MapPanelView) are both JPanels as they are composable panels that allow layouts.
    - Individual “sections” inside each view also used JPanels to encapsulate each section and allow the section to control the layout of its children
    - We used the GridBagLayout for many layouts including the GameView as it allows easy grid-based layout, but still compacting components to fit near and around each other. GridBagLayout also allows components to span multiple columns and rows, which is convenient for GameView where the map view must span three of four columns. We concisudered using the GridLayout, but it does not allow multiple-column and multi-row spanning.
  - JMenuBar/JMenu – we used JMenuBar to create the actual menu bar, and JMenu to implement each menu inside the bar. Each menu contains JMenuItems to represent each button.
  - JDialog – the guess and solve (GuessAndSolveView), and exit confirmation (PromptExitView) are dialog windows so they can be tied to the main application and use dialog standards.
- The interface makes up the ‘View’ component of Model-View-Controller, so the components that used Swing were kept completely separate from the model and controller logic, as to make the model and controller independent of any view system.

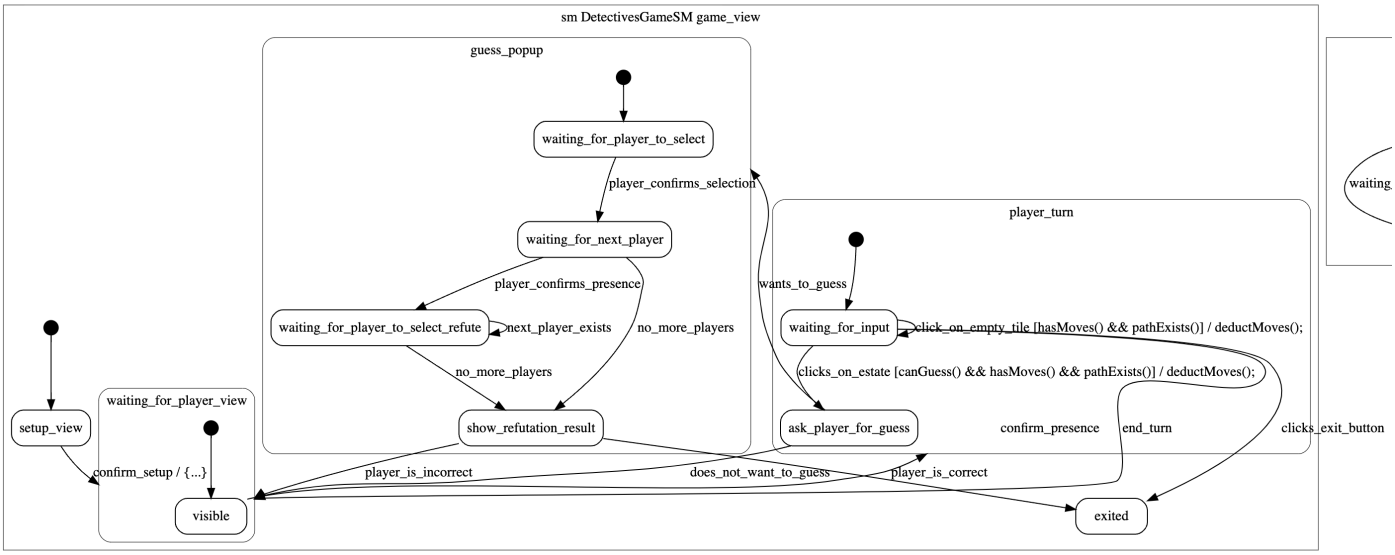
Design

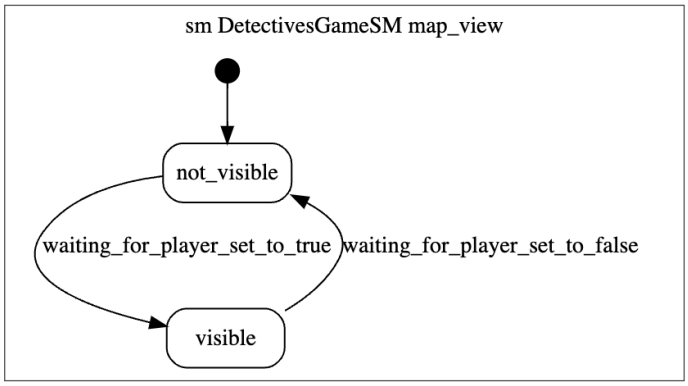
- Our previous design was somewhat supportive of adding the GUI. Our “Game” class acted as both a controller (performing logic) and a model (storing game state), so we decided to split it into GameController (which performs the logic) and GameModel (which stores the state, and notifies listeners when it changes). This allowed us to test GameController and GameModel independently of the view, so our tests do not have to create a JFrame (“headless” operation). In this design, GameController and GameModel are completely oblivious of the UI, and do not use any Swing code. This would allow a developer to “swap out” the UI for a completely different system without impacting the controller and model code.

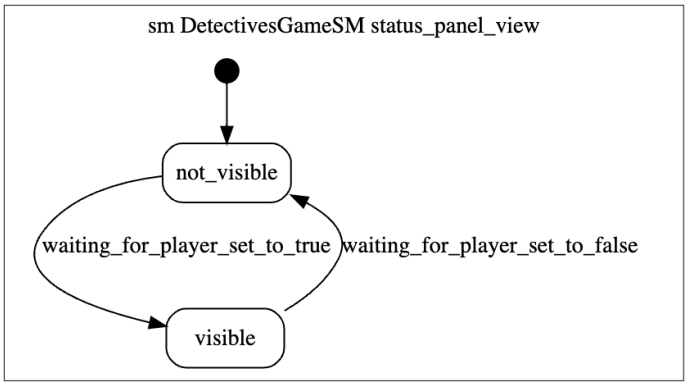


- This required a lot of deletion on the GameModel and GameController, as the textual-based input system is no longer required.
- We also moved from a string of inputs (i.e. “LLUURR” to go left-left-up-up etc) to a path-finding based algorithm, where the player clicks on a tile they wish to go to and an implementation of the A* algorithm finds the shortest path to reach that destination. We had to rewrite large parts of GameController to accommodate this behaviour.
- In the first assignment, we separated the Game class from the Board class, which made it easier to migrate to the MVC model. A lot of the Board logic was independent of the view system, so there were not many changes required to Board, only changes to the Game class.

State diagram







- The major states in the game that affect almost all views are the **waiting_for_player** state and **waiting_for_input** state. The waiting for player state appears after a player ends the turn and before the first turn in the game, and it triggers the Waiting For Player View to render and block the screen until the player confirms their presence. Then, the **confirm_presence** transition sets **waiting_for_player** on the model to false (ending the state) and this triggers the map_view and status_panel_view to render, starting the turn.
- When the game starts, the SetupView listens to the game model and shows itself. When setup is complete, it sends the setup information to the controller, which handles relevant operations, and stores that data in the model. The model then notifies all views that are listening (through PropertyChangeSupport), and the views update themselves accordingly.
- The GameView listens to changes in the model (using PropertyChangeListener) and updates itself, initializes new views, and disposes views accordingly. When the model provides data for the guess popup, the GameView initializes and shows the GuessAndSolveView.
- The map and status panel views both have their own state, which is controlled by updates to the model field “waiting_for_player”. When it is set to false,