Milestone 1: Design

Yuchen Shen (ys393), Yishu Zhang (yz923), and Esther Jun (ejj35)

1 System Description

1.1 Core Vision

An `OCaml` implementation of the popular German board game Settlers of Catan, as well as several bots that may participate in the game.

1.2 Key Features

1. **A graphical user interface** that displays the current game state. A player may use his or her mouse to select graphical icons and thereby play the game.
2. **Permanent structures**. Players may build:
    a. Settlements on empty intersections. Each settlement is worth one victory point and requires one brick, one lumber, one wool, and one grain.
    b. Cities from existing settlements. Each city is worth two victory points and requires three ore and two grain.
    c. Roads on empty edges. Each road requires one brick and one lumber.

    Players may build as many structures as their resource supplies allow. The distance rule requires that every settlement and every city be surrounded by three unoccupied intersections. A settlement must touch a road; both must be built by the player.
3. **Limited resources**. There are five types of resources: lumber, wool, grain, brick, and ore.
4. **Terrain hexes**. A terrain hex (i.e., a hexagonal tile) represents territory and contains six corners where settlements and cities may be built and six edges where roads may be built. Each hex corresponds to a type of resource and is assigned an arbitrary number between two and twelve, with the exception of seven. If a dice roll results in this number, the hex produces one unit of the appropriate resource for each bordering settlement and two units for each bordering city. There are nineteen hexes and six types of hexes:
    a. Forest: Produces lumber.
    b. Pasture: Produces wool.
    c. Field: Produces grain.
    d. Hill: Produces brick.
    e. Mountain: Produces ore.
    f. Desert: Barren; it does not produce resources. Thus, it does not correspond to a dice roll.
    The robber begins in the desert hex.
5. **Development cards** that unlock special events. A development card requires one ore, one wool, and one grain. Players may buy as many development cards as their resource supplies allow. Only one development card may be played per turn; a card may not be played the same turn in which it is bought; once played, a card never returns to the supply. There are twenty-five cards and five types of cards:
    a. Knight: Allows a player to move the robber and increases the size of a player's army.
    b. Road of Building: Grants a player two additional roads.
    c. Year of Plenty: Allows a player to choose any two resource cards from the bank.

    d. Monopoly: Allows a player to steal all the resources of a specific type from the other players.
    e. Victory Point: Grants a player one point.
6. **Special cards**, each of which awards two victory points to its holder. There are two special cards:
    a. Longest Road: The longest road is a continuous road connecting two intersections, which consists of at least five road segments, is not interrupted by game pieces belonging to other players, and has more road segments than any other connecting road of this type. The player with the longest road may claim the Longest Road card and the two points associated with the card.
    b. Largest Army: The largest army is the largest group of knights activated by a single player and must have at least three knights. The player with the largest army may claim the Largest Army card and the two points associated with the card.
7. **A robber** that shakes the game up. The robber strikes if a knight card is activated or a seven is rolled. A series of events occurs, as follows:
    a. If a seven is rolled, every player with more than seven resources must discard half of them, rounded down.
    b. The player whose turn it is may place the robber on any hex.
    c. The player may steal an unknown resource from the hand of a player that has built a settlement or city that borders the hex.
    d. The hex may not produce resources so long as it is occupied by the robber.
8. **Trade** amongst the players and the bank.
    a. Domestic trade. The player whose turn it is may trade resources with the other players; other players may suggest counteroffers or make proposals of their own.
    b. Maritime trade. The player whose turn it is may trade with the bank at a 4:1 ratio. A settlement alongside a harbor allows a player to trade at more favorable rates (e.g., 3:1, 2:1).
9. **A bot** that plays alongside the player.

1.3 Narrative Description

Settlers of Catan is a multiplayer, turn-based board game of resource management and strategic trade. The game may involve anywhere from three to six players, depending on the variation; our implementation will have four players, three of which will be our own bots.

The game board consists of nineteen terrain hexes, which spawn resources, and nine harbors, which provide favorable trade ratios with the bank. A new game generates a new game board, with an arbitrary arrangement of the hexes and an arbitrary assignment of dice rolls to the hexes; harbors remain fixed. In the game setup, the players must place two pairs of a settlement and a road on the board. A settlement may only be placed on an empty intersection; a road may only be placed on an empty edge; and each settlement-road pair must touch. There is also a distance rule requires that every settlement and every city be surrounded by three unoccupied intersections. More detailed instructions may be found in the Catan almanac.

In the main phase of the game, players take turns, during which they:

1. Roll the dice; the result of the dice roll applies to all players. Each terrain hex, with the exception of the desert, that corresponds to the dice roll produces resources for the settlements and cities that border it.

Every settlement receives one of the appropriate resource; every city receives two of the appropriate resource. If a seven is rolled, the robber strikes. The robber scenario is described in detail above.
2. Build a settlement, city, or road. Players may build as many structures as their resource supplies allow.
3. Propose a trade and/or receive a trade offer.
4. Buy and/or play a development card. A card may be played prior to the dice roll; only one may be played. A card may not be played the same turn in which it is bought.

The game ends when a player reaches a certain number of victory points; it is typically ten but may vary according to the number of players.

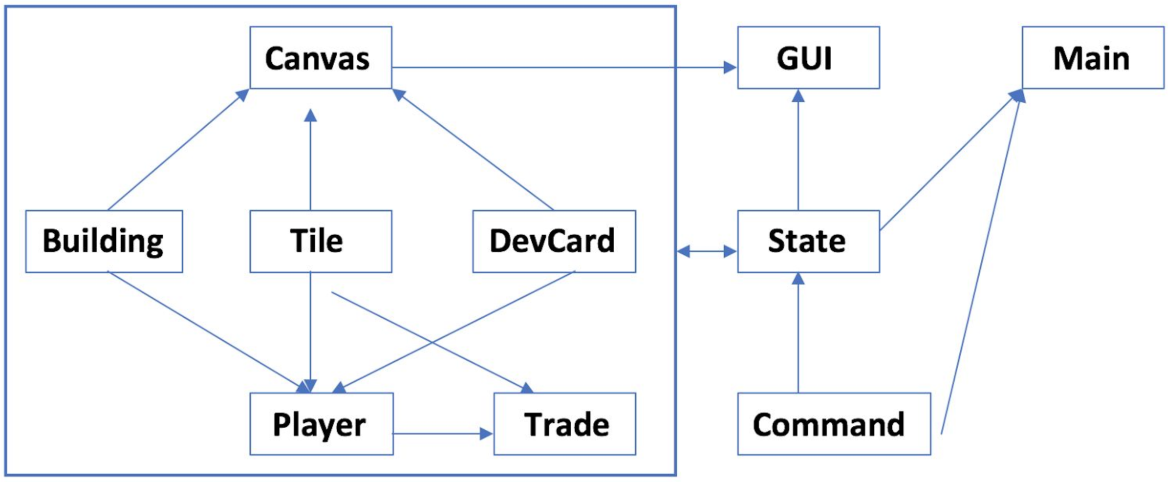
2 System Design

2.1 Key Modules

1. **`Main`**: The module launches the game, handles game mechanics, checks for inputs, interfaces with the GUI, and updates the state at the end of each turn. It is the glue that holds the rest of the modules together.
2. **`State`**: The module contains type `state`, which represents the state of the game, and relevant functions that handle events that modify the state, such as a dice roll, the activation of a development card, and trade. A `state` keeps all the information required to launch and update the game according to different steps, and there is a large do function for player and one for AI, which handles all possibilities during one turn of the player. The state module also links many other modules together and acts as the driving engine for the game.
3. **`Command`**: The module handles inputs from the player (i.e., mouse clicks, keypresses, console input) and translates them into more familiar commands and events.
4. **`Player`**: The module represents a player, which may be a human or a bot, and keeps track of a player's resources, buildings, roads, development cards, trophies, and score. It contains a function that takes in an event and updates player stats accordingly.
5. **`Building`**: The module represents a building (i.e., a settlement, a city, nonexistent). It contains an index that corresponds to location of the building and its settler, if it has one. Also, since different types of buildings correspond to different ratios of resource obtained, this module will keep a multiplier for each building, to show what type of building it is and what ratio of resources will be obtained.
6. **`Canvas`**: The module represents the game board, which consists of hexes and their corresponding buildings bordering each hex terrain.
7. **`Tile`**: The module represents a terrain hex. It contains the resource the hex generates, its coordinates, the corresponding dice roll, the buildings constructed on the hex, and other relevant information. `Tile` provides information necessary to `Canvas` and `Player` and interacts with `State`.
8. **`Trade`**: The module handles trade in resources between two players and between a player and the bank. It contains specific informations about trade ratios.
9. **`DevCard`**: The module represents a development card, which triggers a special event. It provides information of special events for `State` module to handle accordingly.

10. `GUI`: The module shows the current state of the game. More specifically, the GUI displays a menu, player stats, the board, and buildings constructed on the board. Given enough time, we may also include a title screen and a sequence of frames that explains the game's features (i.e., a tutorial).

2.2 Module Dependency Diagram



3 Data

3.1 Storage

We will keep the majority of our data in the `State` module, which contains a type `state` that maintains information about the game, the players, the hexes, and the development cards. `State` will interact with the `Player`, `Tile`, and `DevCard` modules, which hold different record types and variant types, so that it fetches informations from them and is able to utilize utility functions from these parts for information update.

3.2 Communication

We have multiple modules that interact with one another to prompt the gameplay. It is essentially an extension of the Model-View-Controller pattern of game design. A diagram of module interactions is shown above. The basic structures of communications are as follows: `Main` is the module that prompts the REPL for the gameplay, which fetches information from `State` and `Command` modules. `State` fetches information from every other module, apart from `GUI` and `Main`, and updates necessary information for the gameplay. `GUI` fetches information from `Canvas` and `State`. Both `Canvas` and `Player` fetch information from `Building`, `DevCard`, and `Tile`. There are several other helper modules that link the modules together.

3.3 Data Structures and Algorithms

We will use mutable records to store player-related information, settlement-related information and hex-related information because we need to update the ownership of the settlement, the score of the player, etc. We will use records to hold information about the game play state, the cards, the players and the hexes, and we will use variants to represent resources and card types. As for our AI, we plan to use the minimax algorithm to calculate actions and steps and may maintain a `state` tree that is searched by Monte Carlo.

4 External Dependencies

We plan to use the `Graphics`, `Gtk+`, and `Camlimages` libraries to construct and communicate with our GUI. We are also looking into the `OpenGL` and `Xlib` libraries, and we may use the `js_of_ocaml` library if we decide to run the game on a browser.

5 Testing Plan

We plan to do incremental unit testing, integration testing, and system testing.

1. **Unit testing**: We will write black-box and glass-box test suites for important functions in each module; the test cases will cover typical inputs, randomized inputs, and boundary cases. We will also run regression tests to make sure that modifications to one module does not cause strange behavior in another module. We will most likely brush over the game bots, seeing as it will be difficult to write test cases for the AI, whose behavior we cannot predict.
2. **Integration testing**: We will use the top-down approach to test groups of modules—that is, we will test higher-level components before lower-level ones, using stubs as placeholders for unimplemented modules. We may also use a hybrid top-down, bottom-up approach if the circumstances call for it.
3. **System testing**: We will test the gameplay and the bots by having potential users play the game and give feedback; we will look out for strange behavior during these playtests. Needless to say, we will also do playtests of our own to look for bugs and areas for improvement.