# Settlers of Clarktan: Design Document

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

## 1 System Description

### Core Vision

An `OCaml` implementation of the popular German board game Settlers of Catan, which we renamed as Settlers of Clarktan, as well as three AI bots that are competing with the player.

### Description

Settlers of Clarktan is a turn-based board game of resource management and strategic trade. The player will be competing against three bots. In the game players can build settlements, cities and roads to extend their controls over resources, purchase and play development cards, activate the robber to control resource flows, and trade resources with other players, the bank, or the marital ports to their advantages. The player starts off building two roads and two settlements, and the goal is to gain as many victory points as possible. The first player who reaches 10 victory points wins.

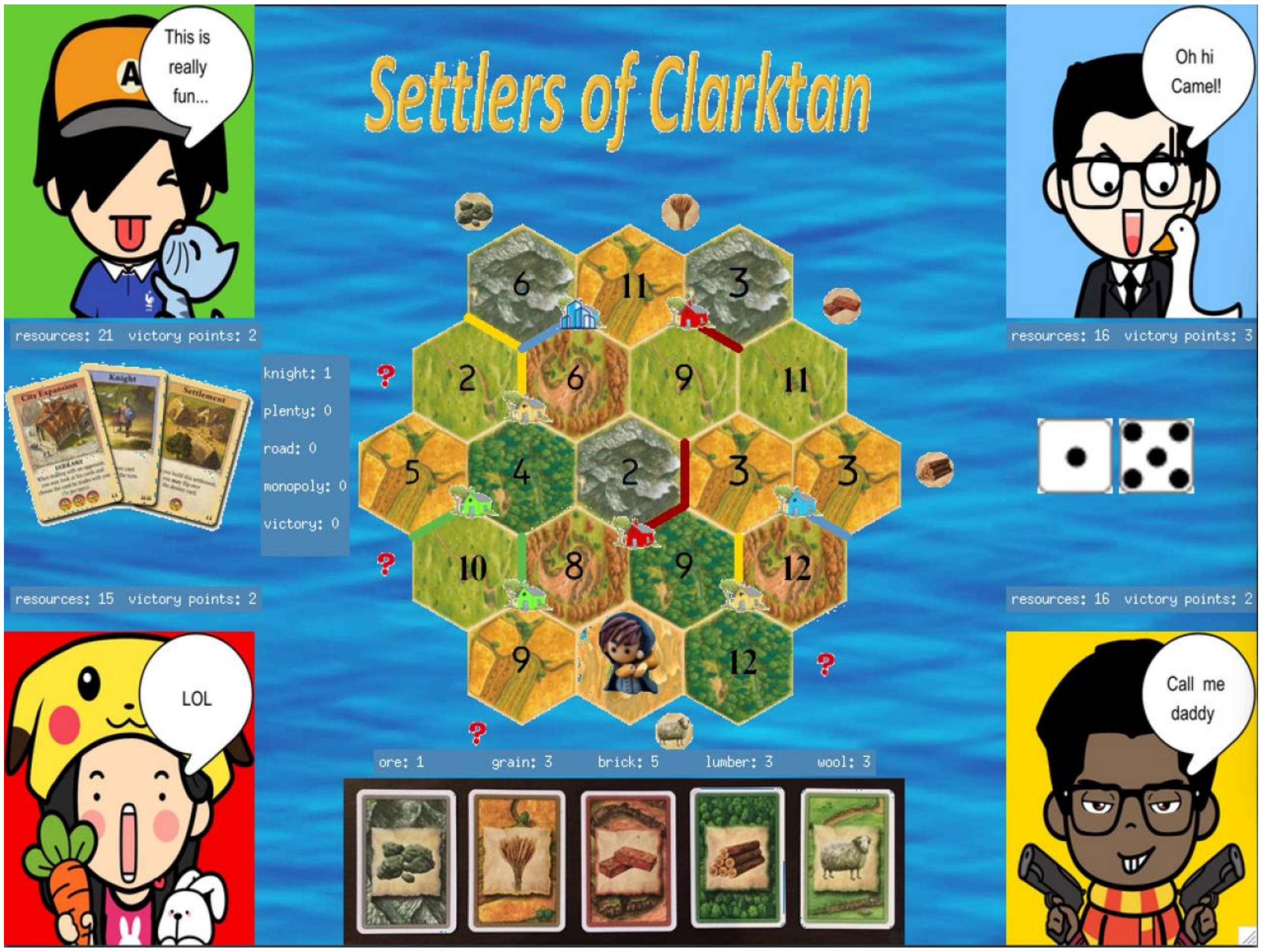
The game board consists of nineteen terrain hexes, which spawn resources, and nine harbors, which provide favorable trade ratios with the bank. A new game generates a new game board, with an arbitrary arrangement of the hexes and an arbitrary assignment of dice rolls to the hexes; harbors remain fixed. In the graphical user interface, the player will be able to observe any changes made to the game board, such as settlements, roads, and cities built by any player in the game, the robber, the dice rolls, and certain information about each player involved in the game.

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, except for 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 in the game rules.
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, trade with other players or the bank.
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.

This is what our final product looks like:



For more extensive explanation of game rules please see:
[http://www.catan.com/en/download/?SoC_rv_Rules_091907.pdf](http://www.catan.com/en/download/?SoC_rv_Rules_091907.pdf)

## Key Features

**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.

**ANSITerminal** that allows player to enter interactive text commands to proceed with the game. Changes to the game board by these commands are reflected through the graphical user interface.

**Permanent structures.** Players may build:
* Settlements on empty intersections. Each settlement is worth one victory point and requires one brick, one lumber, one wool, and one grain.

* Cities from existing settlements. Each city is worth two victory points and requires three ore and two grain.
* Roads on empty edges. Each road requires one brick and one lumber.

Players will have an upper limit on the number of structures they can build in each game, so they must plan wisely. 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.

**Limited resources.** There are five types of resources: lumber, wool, grain, brick, and ore.
The desert terrain produces no resource.

**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, except for 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:
* Field: Produces grain.
* Hill: Produces brick.
* Mountain: Produces ore.
* Forest: Produces lumber.
* Pasture: Produces wool.
* Desert: Barren; it does not produce resources

**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:
* Knight: Allows a player to move the robber and increases the size of a player’s army.
* Road of Building: Grants a player two additional roads.
* Year of Plenty: Allows a player to choose any two resource cards from the bank.

* Monopoly: Allows a player to steal all resources of a specific type from other players.
* Victory Point: Grants a player one point.

**Special cards**, each of which awards two victory points to its holder. There are two special cards:
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.
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.

**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:
* If a seven is rolled, every player with more than seven resources must discard half of them, rounded down.
* The player whose turn it is may place the robber on any hex.
The player may steal an unknown resource from the hand of a player that has built a settlement or city that borders the hex.
The hex may not produce resources so long as it is occupied by the robber.

**Trade** amongst the players and the bank. Trade takes two forms:
* 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.
* 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).

**Several bots** that plays alongside the player.

### Acknowledgement for discrepancy from Settlers of Catan

1. The initial phase of the original game generates resources only for the second settlement the players choose to build, while Settlers of Clarktan generates resources for both. We feel this way the game is easier to play for the players.

2. The original game allows the player to choose from whom he wants to steal a random resource when he activates the robber, while our game has it handled so that instead of letting the player make the choice, we steal a random resource from another player who has a house bordering the robber tile, and give that resource to the player. We made this small change because it has the same effect as with the original game.

## 2. System Design

### Key Modules

**Main:** The module launches the game, handles game mechanics, checks for inputs, interacts with the GUI and the ANSITerminal, and updates the state at the end of each turn. It is the glue that holds the rest of the modules together.

**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 the building of structures, the activation of a development card, and trade. A state keeps all the information regarding the human player required to launch and update the game according to different steps, and there is a large do_move function for player, 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.

**AI:** The module contains all the decision-making rules for the bots, as well as how their decisions is going to affect the state of the game. It also contains a large do_ai function for bots, which handles all possibilities during the turns of the bots, between two turns of the human player.

**GUI:** The GUI module handles all drawings and updates of the game canvas, including the game board of tiles and all sorts of information regarding players and the game system. The GUI basically displays player stats, the board, the robber, and buildings and roads constructed on the board.

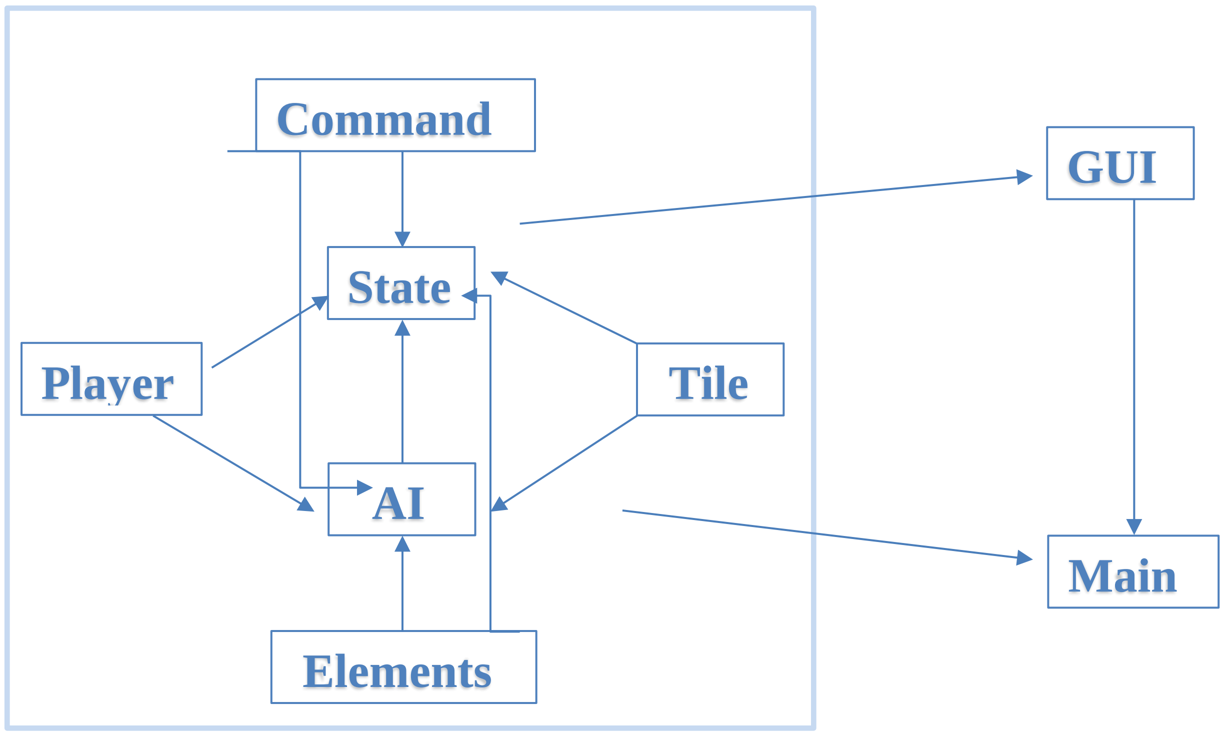
**Command:** The module handles inputs from the player (i.e., mouse clicks, keypresses, console input) and translates them into more familiar commands and events.

**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.

**Tile:** The module represents the 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.

**Elements:** This is the utility module that contains the types for color, which indicates the players in the game, for resource, for development cards, as well as for intersections and edges used in our numbering system for the game board. It also contains several utility functions used throughout our project.

**Module Dependency Diagram**



See the .mli files for a more in depth look at what each module does.

## Data Storage

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

Data are also preserved in the GUI canvas of the game, and some parts of our data storage can be visualized from the player's point of view. Certain changes to the data storage are reflected through the GUI as well.

## 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.

## Data Structures and Algorithms

We use 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 use variants to represent resources and card types. We designed a type of plan tree in AI to calculate the next plan of the AI to calculate better strategies for the bots. We use association lists multiple times throughout the project for searching and updating information when the information list is not exceedingly long.

For algorithms, we use mainly Depth-First-Search in state to implement the longest road token, as well as in AI to help the bots figure out the next plan or strategy. We also took a leaf from the Mini-Max algorithm to design our strategies for bots.

## External Dependencies

Our implementation depends heavily on the Graphics library and the CamlImages library.

**Graphics**: This is used to draw the GUI, record and handle the mouse events. We used Photoshop to help crop and reformat images so that they work with the Graphics module.

**CamlImages**: We used several modules and many functions from here, and they are all vital for our GUI. We load images from a given file in assets folder, and then convert the image to a color array array so that it can be used in Graphics.

## Testing

We did incremental unit testing, integration testing, and interactive system testing.

1. **Unit testing**: We wrote black-box and glass-box test suites for important functions in each module; the test cases extensively cover typical inputs, randomized inputs, and boundary cases. We also ran regression tests to make sure that modifications to one module does not cause strange behavior in another module.
2. **Integration testing**: We used the top-down approach to test groups of modules—that is, we test higher-level components before lower-level ones, using stubs as placeholders for unimplemented modules. We also used a hybrid top-down, bottom-up approach when the circumstances call for it.
3. **System testing**: We did extensive interactive testing using the Graphics canvas and the ANSI Terminal. We tested the gameplay and the bots by having potential users play the game and give feedback; we looked out for strange behavior during these playtests.

## Division of Labor

**Yuchen** did the drawing and updating part for the Graphical user interface, the structure building and robber parts in State and AI, as well as interactive testing. This included designing the graphics, drawing the image assets to the screen, updating the game canvas according to game progresses, handling the back-end structure building parts for human players (which includes road, settlement, and city buildings), dealing with robber events, and handling the decision plan of bots. She spent around 60 hours' total on the project, including finding, editing, and creating images and font, coding up the parts in State and AI, and interactive play testing.

**Esther** did the underlying indexing system that supports the GUI and game board, the handling of player text inputs and mouse events, the main loop, and the development card and inter-player trading parts of State and AI, as well as interactive testing. This includes designing the numbering system for the hexagon terrain board, dealing with command of both text and mouse click formats, making the main loop run properly for the game play, handling the back-end card playing parts for human player and bots, and ensuring trading between players and bots work for the game. She spent around 60 hours' total on the project, including coding up the supporting features, the parts in State and AI, the main loop and command, and interactive play testing.

**Yishu** did the unit testing part of the game, the trading with bank and ports parts in State and AI, and the longest-road algorithms in State. This included designing unit testing suits for black-box and glass-box testing of the game’s important modules, coding up the related trading parts for State and AI, manipulating the longest-road algorithm using Depth-First Search, and making some other miscellaneous utility functions used throughout the projects. She spent around 50 hours’ total on the project, including coding up the related parts in State and AI, as well as the unit test suits.

## Known issues:

Not to our awareness.