WARS OF CATAN

Authors:
Ankit Bhutani,
Paul Graff,
Kyle Grage,
Marty Malecki,
Alex Politakis,
Cody Schuffelen

Table of Contents:

1.1: What is Wars of Catan:
1.2: Tools:
2.0: Our Team Process
2.0.1: Scrum vs Extreme Programming
2.02: Development Process
2.0.3: User Story Definitions
2.0.4: Story Progress Definitions
2.1: Scrum Team Meetings
2.1.2: MARS roles -> Scrum Roles
2.1.4: Story Workshop
2.1.5: Daily Stand-Up
2.2: Scrum Analysis
2.2.1: Iterative Development
2.2.2: Refactoring
2.2.3: Testing
3.0: Requirements
4.1: Model
4.2: View
4.2.1 The Game Board
4.3: Controller
4.3.1: Event Handlers
4.3.2: Control States
5.0: Installation
5.1: On Ubuntu 12.04 LTS:
5.2: on Arch Linux
6.0 Future plans
7.0 Personal Reflections

1.1: What is Wars of Catan:

Wars of Catan is a computer game based off of Settlers of Catan, a popular board game by Mayfair Games. The goal of each player is to gain victory points by placing settlements, cities, roads, and playing development cards, and the first player to get ten victory points, wins. Since the game is set up randomly, no player is guaranteed to have access to all the resources needed to win. Players will need to trade with one another in order to get the resources they need, but the challenge is that the motivation behind all these deals is left to the players themselves. Players must forge alliances, steal, lie, and compete against one another to get enough resources to reach the top and win the Wars of Catan.

1.2: Tools:

Wars of Catan is written in C++ using the graphics library framework SDL, OpenGL, and the SDL extension SDL_ttf. SDL is a library that allows the developer to use the graphics library OpenGL more easily. More information on SDL can be found at http://www.libsdl.org/. SDL has several extensions, such as SDL_ttf which we used for rendering text (found at https://www.libsdl.org/projects/SDL_ttf/). For testing we used the unit testing framework Google Test. More information on that can be found at https://code.google.com/p/googletest/.The code is hosted on GitHub.com at the address https://github.com/Databean/warsofcatan. Anyone wishing to expand on our project should refer to section 2.0 on our process before creating a pull request.

2.0: Our Team Process

Our team decided to make use of Agile-Scrum (referred to as Scrum).

2.0.1: Scrum vs Extreme Programming

The main differences between Scrum and Extreme Programming is that Scrum puts more emphasis on planning, but trusts the developers to develop in the best way they can and trusts the developers to automatically write clear code and clear documentation.

2.02: Development Process

Development will be done by individuals according to a story and, if available, any high-level designs provided by the team. The developer(s) responsible for a story must divide the story into subtasks on the Scrum Board and are responsible for keeping the progress of all subtasks updated. Subtasks can be added at any time and as needed. All completed stories must be tested by someone serving a Quality Analysis role.

2.0.3: User Story Definitions

1. **Epic**: A large overall concept to develop for
2. **Story**: The smallest form of functionality such that completion will result in something of value.
3. **Subtask**: Any unit of workload towards developing a story such that only one person should be responsible for completing the specified work.

2.0.4: Story Progress Definitions

To Do: No work towards completing this task or story exists.

In Progress: Work towards completing this task or story exists.

Done: The story or task requires no further work and/or testing.

Blocked: When progress towards this story or task cannot be furthered until another story or task is complete.

Cancelled: When no further work should be pursued towards this story or task.

2.1: Scrum Team Meetings

2.1.1: Sprint Review

These meetings are equivalent to Iteration meetings where all team members review the work done and present plans for the future. The Sprint Review follows the format outlined in the Sprint Planning/Retrospective.

2.1.2: MARS roles -> Scrum Roles

(Customer -> Product Owner): This person is the person mandating requirements for a story. (All members will serve this role as we are developing for ourselves).

Moderator -> Scrum Master: Similar to the moderator, the Scrum Master keeps track of the scrum process and uses this to guide meetings.

Scribe -> Manager: This person is in charge of keeping track of meetings, synchronizing the team, and making sure all standards are followed.

Reviewers -> Quality Analysts: In Scrum, this person reviews code via tests and compares it to a story's acceptance criteria. This person will usually be in charge of demoing manual test plans and automatic tests.

Authors -> Developers: These are the people writing the core code.

2.1.3: Sprint Planning/Retrospective

This meeting is done before the Sprint Review meeting. The team evaluates how well this sprint went as well as work that is done, new work that must be done, and problems/solutions for the sprint.

2.1.4: Story Workshop

This is where all team members decide on which backlog stories will enter the next Sprint by either grabbing a story from the backlog, or creating a new story. This meeting should also be used to reprioritize the backlog and create or delete stories based on the team's current development status.

2.1.5: Daily Stand-Up

This is a meeting where developers get together and discuss what they have done, what they are planning on doing / their goals, anything that may be stopping them.

2.16: Modification of Stand-Up

Our team is not a corporation. Everybody has differing schedules instead of the same 8-hour schedule. Furthermore, we do not mandate that all work is done in the same place (everybody can work from home). In order to convey similar standards to the Daily Stand-Up, all group members will be required to maintain the scrum board in real-time relative to their work. Additionally, our weekly team meetings will serve as a weekly stand-up and follow the same format as a Daily Stand-Up.

2.2: Scrum Analysis

2.2.1: Iterative Development

In terms of iterative development, our entire team felt that Scrum was a better process than XP. The most useful part of Scrum was our Scrum board, where we can easily see what stories are assigned to us and anything that may be blocking a team member's work.

2.2.2: Refactoring

In general, Scrum has nothing to say about refactoring, but this is actually a good thing. Our team agreed to a few coding standards and in general, we refactored when we saw a need to. We also made use of pull requests in GitHub so other developers may check for confusing code.

2.2.3: Testing

Scrum again says nothing about how to test except that there should be tests. Our team wrote a few tests and in the beginning, we struggled to write thorough test cases. When we realized that we were lacking this, we were able to evaluate the situation using Scrum's Retrospective feature, and then added a "keep the lights on" (Technical Debt) story to fix it. We originally started with UnitTest++ but switched over to Google Tests due to the class requirement that we needed parameterized testing.

2.2.4: Collaboration

Collaboration was probably the biggest plus (compared to XP) that Scrum and GitHub had. We updated our Scrum board as we completed our sub-tasks and stories. Our Scrum board was great for getting a quick overview of everything going on within the team. Furthermore, it was great using GitHub branches in order to better track each feature and allow us to separate our programming.

2.2.5: Miscellaneous

Our retrospectives provided us with good guidance about what was working and what was not working. Overall, we never had a perfect Sprint and always seemed to have something we can improve on. The process was often hindered by our differing schedules.

3.0: Requirements

Requirements function in this project both as specifications for the software and as rules for the game. We've listed use cases and user stories completed to mark the implemented portions of Wars of Catan.

3.1: Use Cases

- **Users must take turns while playing the game:** No two players can be making moves at the same time. Players must take turns instead. The player who taking his turn is defined as the current player, and the other players are called opposing players at this point. The game must indicate who is the current player.
- **User must be able to trade with other player**: At any point during his turn, after the dice has been rolled, the current player can offer a trade to any other player in the game. If the other player accepts the trade, the resources are exchanged.
- **User must be able to trade with the bank**: At any point in a turn, the current player can trade 4 of one kind of resource card that he possesses for another type of resource card.
- **User must be able to build a Settlement**: If the current player has one Wood, Sheep, Wheat, and Brick, they can buy a Settlement using those resources. The current player should then be able to place that settlement on a vertex on the board.
- **User must be able to build a City:** If the current player has three Ore and two Wheat, he can buy a City using those resources. The current player can then upgrade one of his settlements into that city they bought.
- **User must be able to build a Wonder:** If the current player has five of each resource, he can buy a Wonder using those resources and place it over one of his current settlements or cities. The Wonder will cause the current player to win the game.
- **User must be able to acquire five different types of resources**: Users will accumulate resources of type Wool, Wheat, Ore, Lumber, and Brick.
- **User must accumulate resources according to his properties, board tile values, and the position of the robber**: Upon rolling of the dice, a user receives one resource for each settlement he controls adjacent to a similarly-numbered resource, and two resources for each adjacent city. Resources will not be accumulated from tiles currently covered with the robber piece.
- **Resource information must be available exclusively to the user that controls those resources**: The current player only has access to the resource totals constituting his hand, and not the resources constituting an opposing player's hand.

- **User must be able to build a Road**: If the current player has one Brick and one Wood, he can buy a road using those resources. The current player can then place that road between two vertices on the board, off of a settlement, city, or other road.
- **When a player rolls a 7, they should have the option of moving the robber**: Upon the beginning of a turn, the game will roll the dice automatically. The user moves the robber if a 7 is rolled.
- **User must be able to buy a Development Card:** If the current player has one Wool, Wheat, and Ore, then he can buy a development card. A random development card is drawn from the deck and added to the player's current hand.
- **User must be able to play a Knight Development Card:** The current player can buy a development card from the deck. If the card is a Knight, then the user can play this card which will allow him to move the robber to a different location and steal a random resource from an opposing player near the robber's new location.
- **User must be able to gather Victory Point Cards:** The current player can buy a development card from the deck. If the card is a Victory Point card, the current player will gain one additional victory point.
- **User must be able to play a Monopoly Card:** The current player can buy a development card from the deck. If the card is a Monopoly Card, the player can play the card which will allow him to steal all of one resource from all opposing players.
- **User must be able to play a Year of Plenty Card:** The current player can buy a development card from the deck. If the card is a Year of Plenty Card, the player can play the card which will give him two of one resource.
- **User must be able to play a Road Building Card:** The current player can buy a development card from the deck. If the card is a Road Building Card, the player can play the card which allow him to build two roads at the same time for free.

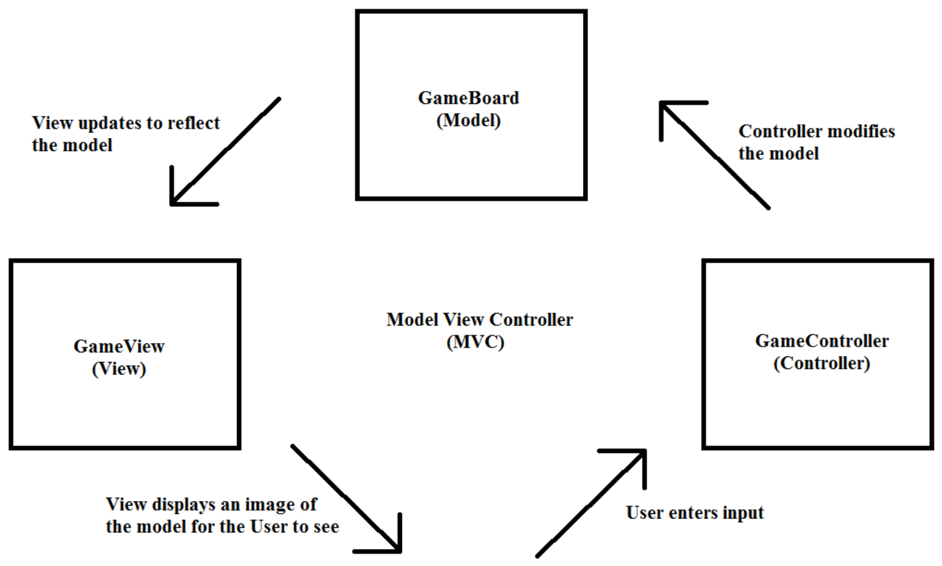
3.2: User Stories

- As a developer I want to set up a github repository and ensure all members of the team can pull and push without error, so that we can share code effectively.
- As a developer I want to create a board with checkers pieces that can be moved around without game piece logic, so that I can understand how the SDL framework is used.
- As a developer I want to create tests for the player class in order to have checks for functionality and regression.
- As a developer, I want to create data structure representation of board in order to set up a foundation for Wars of Catan
- As a developer, I want to create a random board generator in order to get a randomized gaming experience

- As a developer, I want to create a preset board generator in order to load/save a board from/to file
- As a developer, I want to Render a generic Catan board with standard tiles in order to provide a visual of the board
- As a developer, I want to outline resource cards, development cards, settlements, cities, roads, and victory points in order to set a foundation for implementation
- As a developer, I want to implement a deck of developerelopment Cards
- As a developer, I want to implement Settlements and Cities
- As a developer, I want to Implement Roads
- As a developer, I want to Implement Victory Points
- As a developer, I want to Implement trading
- As a developer, I want to Implement robber
- As a developer, I want to Implement dice rolling
- As a developer, I want to implement buying and using development cards
- As a developer, I want to implement buying and placing settlements/cities
- As a developer, I want to implement buying and placing roads
- As a player, I want to click on buttons and the map to buy and place settlements and roads
- As a player, I want to see a display of my current resources
- As a player, I want to see a display of what number was rolled on my turn
- As a player, I want a UI that allows me to see and play development cards
- As a player, I want to click on a button to trade with other players and the bank
- As a player, I want to click a button to end my turn, and a visual indicator of whose turn it is
- As a player, I would like to be able to see and place the robber in the game
- As a player, I would like to be able to see and place the wonder in the game
- As a player, I would like a visual indicator of whose turn it is
- As a player I would like the ability to place roads and settlements as well as have a visual indicator of how many are available to be placed
- As a player, I would like to be able to purchase an object (card, settlement, city, road) in the game
- As a developer, I would like players to randomly lose resources when the robber is rolled (in accordance with game rules)
- As a player, I would like a visual representation of my stellar victory
- As a player, I want the game to open ready for 4 players to start playing

4.0: Architecture & Design

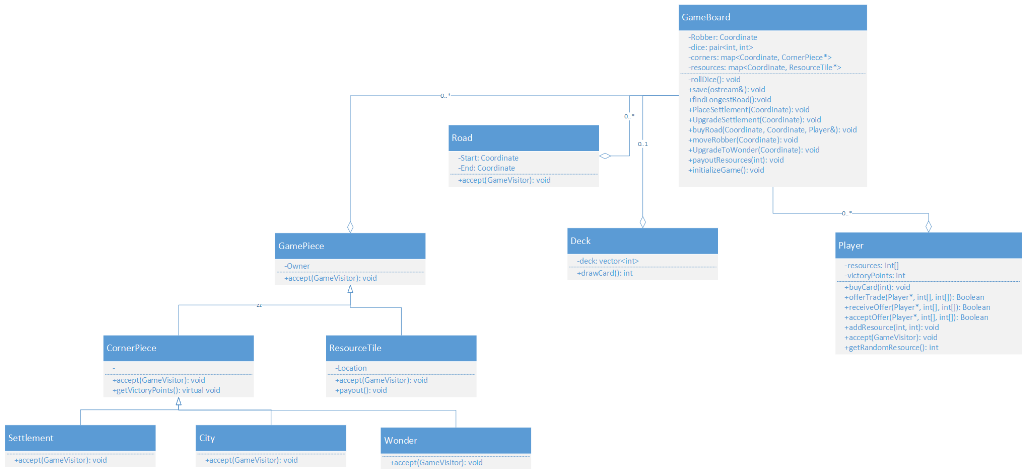
The architecture for Wars of Catan follows the Model View Controller (MVC) pattern due to the heavy reliance of UI because this is a game. The MVC pattern works by separating the program into three main parts. The model, the view, and the controller. The model is essentially the backend architecture of the program. It is the exact state that the program is currently in. The view is a sort of window for the user to see the model. It is the graphics that are being displayed to the screen, but it is designed to simply project an image of the model. The controller is then the vehicle in which the user can interact with the program. It contains functions to handle user input and modify the model based on that input.
figure 1: the MVC pattern


GameBoard
(Model)
View updates to reflect the model
Controller modifies the model
GameView
(View)
Model View Controller
(MVC)
GameController
(Controller)
View displays an image of the model for the User to see
User enters input

4.1: Model

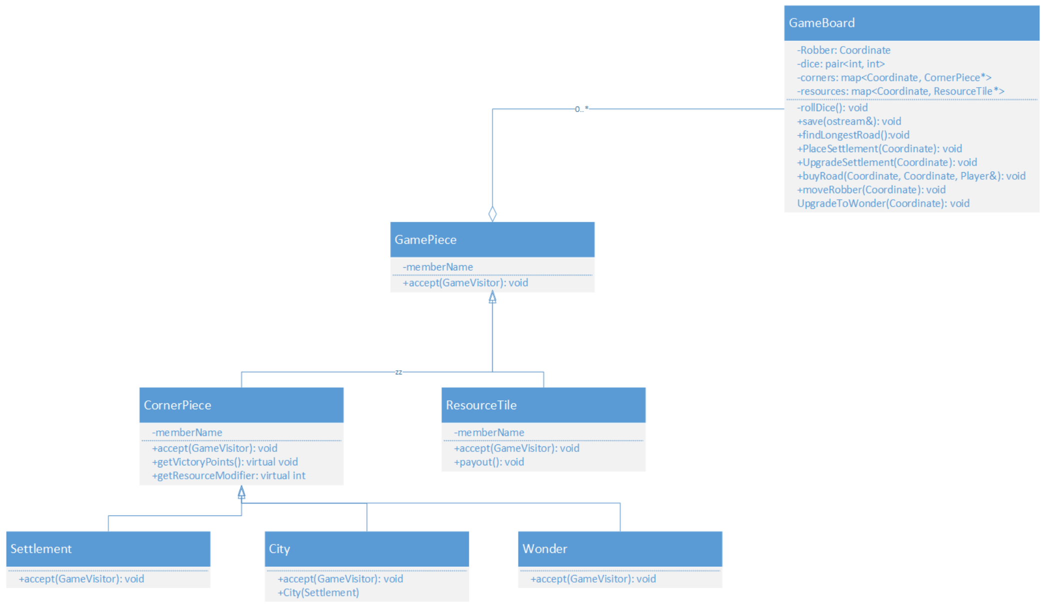
The Game Board serves as a launching point for most of our critical model functionality. The first UML diagram represents the highest level view of our model, with most of the functionality accounted for. The subsequent diagrams serve to highlight finer grained functionality within a particular family of use cases.

figure 3: Full Model


GameBoard
-Robber: Coordinate
-dice: pair<int, int>
-corners: map<Coordinate, CornerPiece*>
-resources: map<Coordinate, ResourceTile*>
-rollDice(): void
+save(ostream&): void
+findLongestRoad():void
+PlaceSettlement(Coordinate): void
+UpgradeSettlement(Coordinate): void
+buyRoad(Coordinate, Coordinate, Player&): void
+moveRobber(Coordinate): void
+UpgradeToWonder(Coordinate): void
+payoutResources(int): void
+initializeGame(): void
0..*
0..*
0..1
0..*
Road
-Start: Coordinate
-End: Coordinate
+accept(GameVisitor): void
Deck
-deck: vector<int>
+drawCard(): int
Player
-resources: int[]
-victoryPoints: int
+buyCard(int): void
+offerTrade(Player*, int[], int[]): Boolean
+receiveOffer(Player*, int[], int[]): Boolean
+acceptOffer(Player*, int[], int[]): Boolean
+addResource(int, int): void
+accept(GameVisitor): void
+getRandomResource(): int
GamePiece
-Owner
+accept(GameVisitor): void
zz
CornerPiece
-
+accept(GameVisitor): void
+getVictoryPoints(): virtual void
ResourceTile
-Location
+accept(GameVisitor): void
+payout(): void
Settlement
+accept(GameVisitor): void
City
+accept(GameVisitor): void
Wonder
+accept(GameVisitor): void

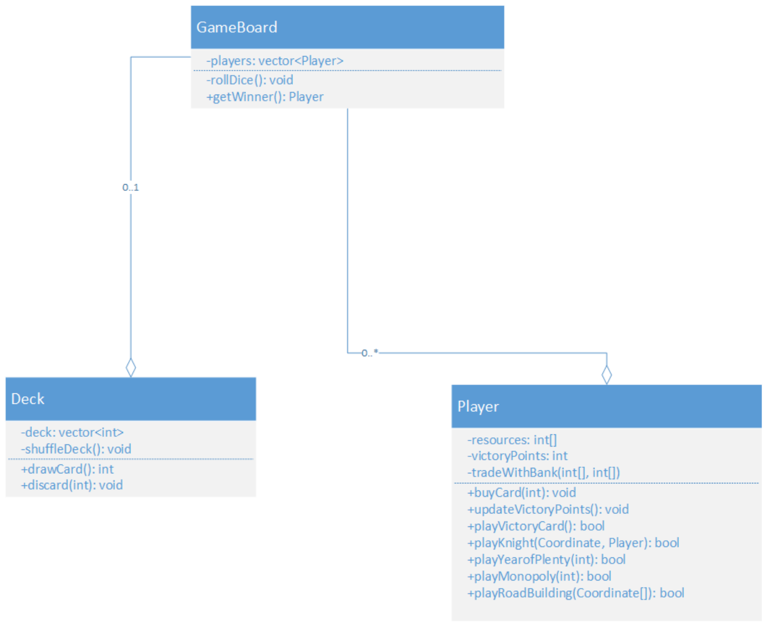
The Gameboard is an unapolagetic god class, but we felt this architecture was allowable for a board game.nearly all of our user stories invoke the usage of this class on some level.

figure 4: Tiles and Pieces


GameBoard
-Robber: Coordinate
-dice: pair<int, int>
-corners: map<Coordinate, CornerPiece*>
-resources: map<Coordinate, ResourceTile*>
-rollDice(): void
+save(ostream&): void
+findLongestRoad():void
+PlaceSettlement(Coordinate): void
+UpgradeSettlement(Coordinate): void
+buyRoad(Coordinate, Coordinate, Player&): void
+moveRobber(Coordinate): void
UpgradeToWonder(Coordinate): void
0..*
GamePiece
-memberName
+accept(GameVisitor): void
zz
CornerPiece
-memberName
+accept(GameVisitor): void
+getVictoryPoints(): virtual void
+getResourceModifier: virtual int
ResourceTile
-memberName
+accept(GameVisitor): void
+payout(): void
Settlement
+accept(GameVisitor): void
City
+accept(GameVisitor): void
+City(Settlement)
Wonder
+accept(GameVisitor): void

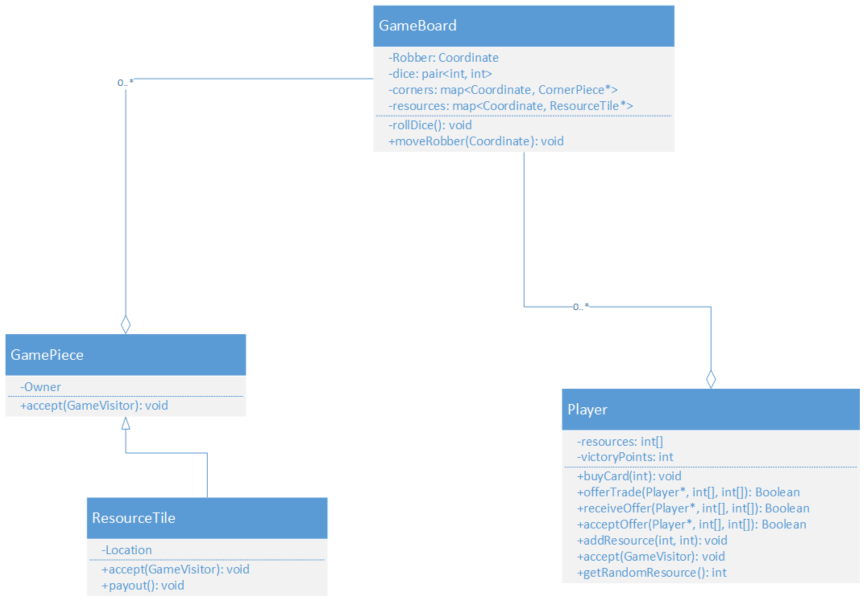
The settlement, city, and wonder functionality constitute the core of the backend. every turn invokes a dice roll, and every dice roll invokes payout methods in this part of the model.

figure 5 Player and Cards



Players interact with the deck of both resource and development cars, here abstracted into one metaphorical deck. Development cards are purchased and played from the player class.

figure 6 Players and Resources


GameBoard
-Robber: Coordinate
-dice: pair<int, int>
-corners: map<Coordinate, CornerPiece*>
-resources: map<Coordinate, ResourceTile*>
-rollDice(): void
+moveRobber(Coordinate): void
0..*
0..*
GamePiece
-Owner
+accept(GameVisitor): void
ResourceTile
-Location
+accept(GameVisitor): void
+payout(): void
Player
-resources: int[]
-victoryPoints: int
+buyCard(int): void
+offerTrade(Player*, int[], int[]): Boolean
+receiveOffer(Player*, int[], int[]): Boolean
+acceptOffer(Player*, int[], int[]): Boolean
+addResource(int, int): void
+accept(GameVisitor): void
+getRandomResource(): int

The robber accomplishes several things: it affects resource accumulation by players, and represents a transfer of resources via stealing functionality

4.2: View

The View is the front end of the game consisting of the actual playing board and a bunch of buttons.

4.2.1 The Game Board

The Game Board is made up of 19 Resource tiles, which can be one of the 5 resources or a dessert. Each resource is hexagon in shape. Each vertex of a hexagon is point where a city, settlement or a wonder can be built. Also a road can be built between two adjacent vertices. A city is represented by a diamond. A city is represented by a square. A wonder is represented by a very large square. A road is just a line between the two points.

4.2.2: The Buttons

On the top right there is a list of names of all the players. Clicking on any one of them initiates a trade dialogue with that player. On the middle right you can see a list off all the resources the player has. On the bottom right there is a list of all the developement cards and a counter next to each to show how many the player holds. There is a button to hide all the resources to prevent other players from peeking over your shoulder. On the bottom left there are also 4 buttons. Three are building roads, cities and settlements. The last one is to end turn.

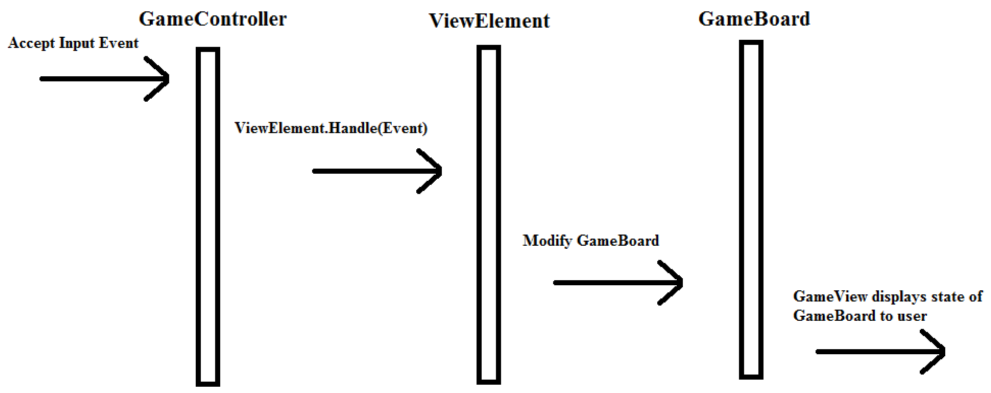
4.3: Controller

The controller is the vehicle through which the user can interact with the model. In the case of Wars of Catan, the controller is the class GameController and the model is the class GameBoard. GameBoard is explained above in section 4.1. GameController functions through a combination mechanisms called Event Handlers and Control States.

4.3.1: Event Handlers

Event Handlers are functions which are bound to UI elements and are called when the user interacts with those UI elements. In our program those UI elements were defined in the class ViewElement. Whenever user input was registered by SDL (a graphics framework based off of OpenGL), the information from that input would then be passed through each ViewElement. We typically call those instances of user input “event.” If the ViewElement recognizes that the event was meant for itself, then it will run its Event Handler to respond to that event. If the Event Handlers is successful then we consider the event handled and prevent other ViewElements from responding to that event. Event Handlers were designed to make modifications to the model (GameBoard) based on the current Control State, and the results of each interaction with a ViewElement can change significantly from each state to the next.

figure 2 How Inputs are Handled


GameController
Accept Input Event
ViewElement
ViewElement.Handle(Event)
GameBoard
Modify GameBoard
GameView displays state of GameBoard to user

4.3.2: Control States

The way events are handled change based on the current Control State of the program. The Control State mainly dictates whether or not many ViewElements are able to handle event and simple causes the elements to not affect the model. In some cases though, the Control State can change the way the ViewElement modifies the GameBoard.

5.0: Installation

5.1: On Ubuntu 12.04 LTS:

To install: Run these commands in order. Make sure you have git installed

- git clone https://github.com/Databean/warsofcatan.git
- sudo add-apt-repository ppa:antumdeluge/sdl2 -y
- sudo apt-get update -y
- sudo apt-get install libsdl2 libsdl2-dev libsdl2-ttf libsdl2-ttf-dev -y

To build and run: Run these commands in order

- make
- ./warsofcatan

To install the testing library: Run these commands in order

- sudo apt-get install libgtest-dev
- cd /usr/src/gtest && sudo cmake . && sudo make && sudo mv libg* /usr/lib/

To build and run the tests: Run this command

- make tests

5.2: on Arch Linux

To install: Run this command

- pacman -Syu sdl2 sdl2_ttf gtest

To build and run: Run these commands in order

- git clone https://github.com/Databean/warsofcatan.git
- ./warsofcatan
- make

To run the tests: Run this command

- make tests

6.0 Future plans

We plan to leave our project up on github as open source software that may be edited again some day.

7.0 Personal Reflections

- Paul
    - I think our process was incredibly successful. By far the best way to organize a team I've ever used. Albeit, I haven't been exposed to much else. I think communication between members was a huge problem though. It was very difficult to get everyone in the same room, and by the time we could get together, we'd have lost a lot of time in the iteration.
    - I liked the project, but I was disappointed that we could never get to the creative point we wanted to get to. I realize I could have put more hours in, but at the same time I don't think that would have been enough. I wish I had had more time to commit to this project.
    - One small final note C++ and OpenGL are the wrong tools for writing a board game. Languages like Python or Java are much better suited for this. Or even a game engine would have been nice to use.
- Marty
    - The project was certainly an interesting one. I feel accomplished to have been a part of this project, yet I think I would change a few things in future projects. While I did enjoy working with OpenGL, I think that in the future I would use a framework or game engine more suited to creating a board game like this. I think that would help us to write more readable and maintainable code as well as have more predictable GUI behavior.
    - The process we used (Scrum) was about the best possible process to use. It allowed us the flexibility we needed to work as a team as well as maintain our commitments to other classes and activities. The only downfall I think we had

    was in our communication - sometimes work would be duplicated because members did not adequately communicate the progress they were making.
  - I don't view this class as being valuable. I think it was a lot of busy work in a poorly structured class that was quite frankly a waste of all of our time. This project taught me about another method of programming in a team, however I think I will still need to learn a new software development style when I am working in the real world (unless they use our exact same style which is unlikely). I think that a university of our caliber should be ashamed to offer a course that is this poorly structured and that has little value outside of itself.
- Kyle
  - This project was a very good learning experience. Before this project, 5/6 of us did not have any experience with C++ graphics (OpenGL and SDL). We ended up making a semi-replication of Catan. Our group had some good work on the front end, but started to write harder-to-read code as we started diving into the GUI. On another thought, I really wish we worked with a common IDE, which would have taken away from the re-setup time when my hard drive failed and allowed me to work better with my group during that time.
  - I could not have thought of a better process to use than Scrum. In reality, all corporate processes will break down at some point because of everybody's other commitments. Overall, those other university commitments were the only thing clouding the process.
  - Overall, this class will always have the handicap that it is one of many University classes. None of these processes will truly work as they would in real life. Furthermore, I did like our freedom to choose our process, but the iteration meeting requirements severely favored those who stuck with XP (like the refactoring requirement, what to test, how many stories had to be completed, and so on).
- Alex
  - I wish we had all been able to get Visual Studio working, as I would have loved some experience with that.
  - I feel my C++ coding maturity has greatly improved from where it was previously (from 225)
  - Exposure to a new Development methodology (Scrum) was interesting and a valuable learning experience
  - Things often didn't go according to plan, but I thought that was actually a good thing. It exposed us to some more difficult aspects of group work, and we weren't penalized to severely for it

    - I really wish we were able to segregate duties more finely. I think CS 429 could be a really cool class if you were able to have more dedicated roles, i.e. architect, designer, UI dev, backend dev, etc. I do understand why the course struggles to organize itself that way, and it's much easier from an evaluation standpoint to require code from everyone.
- Ankit
    - The project was very exciting as I got to learn new technologies and develop a game that I would like to play. Most of the team did not have any experience with the libraries we used(OpenGL and SDL) while starting out. But we still managed to make a really good version of Catan for the PC. I do wish we had all worked on the same IDE which would have made some things a lot easier.
    - Scrum was possibly the best process we could have used. It allowed us the flexibility we all needed to work on a project while balancing other classes. I do feel we needed to have better communication. I think after a whole semester of working with this project, I have really caught up again with C++(I had forgotten a lot since 225) and also learned a lot of new things about the language.
    - This class gave us some experience on how coding in a team would be like in the real world. But due to everyone having a lot work for different classes, we could not get the same experience as working in a company would have given us. Also since we physically meant only once a week, there was a slight communication gap.
- Cody
    - Ambivalent about SCRUM. It didn't take much time to update, but I don't feel like I got anything from it either. Maybe if the project was big enough that I couldn't easily read everyone else's changes.
    - I attempted to use the "build the simplest thing that works" policy to avoid speculative generality, but in some cases it resulted in serious technical debt.
    - The combination of use of cutting-edge C++11 features and varying platforms for the developers and the build server didn't go well.
    - This project has the most code for any group project I've worked in. It was a new experience to get a continuous stream of new code to read and understand and work with, even if I wanted to change it.