CSCI-466
CMP
Project 3
Bapar.5gbfree.com

November 27, 2017

Carlos Perez
Brent Parker

Contributions:

Mathew Griffin:

System Requirements and Assisted with Glossary (report 1)

Stakeholders, Actors & Goals, Casual Use Cases, Traceability Matrix, Use Case Diagram (report 1)

Carlos Perez:

Project Management, assign tasks, table of content. (report 1)

User Interface Specifications, Preliminary design, User effort estimation. (report 1)

Revised project management, user interface specifications. (report 1)

Interaction diagram (sd DominionPlayGame), , integration and testing (report 2)

Class Diagrams, history of work (future work). (report 3)

UI Design and Implementation, Work History

Brent Parker:

cover page, Problem Statement, and Glossary of Terms sections. (report 1)

Sequence Diagram, added terms to glossary. (report 1)

Revised Problem Statement, and Sequence Diagram (report 1)

Cover page, contents, select opponent and view high scores sequence diagrams. (report 2)

Data Structures , class diagrams, data types and operation signatures (report 2)

Effort estimation (with use case points), history of work. (report 3)

Design Patterns, Object Constraint Language Contracts

Table of Contents

Problem Statement

The goal of this project is to create a video game version of the card game, Dominion. This game will run in a web browser so that users can simply go to a website and start playing. The initial version doesn't need to have all the features that the physical card game offers, but more features can be added to future versions, and expansion packs. In future implementations of the game, we'd like to see multiplayer capabilities, a scoreboard that can be seen by all players, the ability for players to play against the computer, and a log to keep track of players actions. We'd also like to offer expansion packs for the players so that they can add additional cards to the game.

A game of Dominion includes 10 kingdom cards that are chosen at random from a deck of 17 different kingdom cards. 10 copies of the 10 chosen kingdom cards are added to the supply which will be shared by all players. The initial version of this project will offer 11 of the 17 kingdom cards so that players will still get variability in their games. In addition to those cards, three types of treasure cards and four types of victory point cards will also be added to the supply.

Players will each get their own draw deck, hand of cards, area for their played cards, and discard pile. They will start with 7 copper cards, and 3 estates cards. Treasure cards can be played to purchase things. Each type of treasure card is worth a different amount of coins. The types of treasure cards are: Copper, Silver, and Gold, they are part of the supply. Copper is free to buy. Silver costs 3 coins to buy and increases coins by 2 when played. Gold costs 6 coins to buy and increases coins by 3 when played.

At the beginning of the game and at the end of the player's turn, five cards are moved from the draw deck to the player's hand. If a player attempts to draw a card when no cards remain in the draw

deck, then all the cards from the discard pile are moved to the draw deck and shuffled to randomize the order.

The player's hand contains the cards drawn from the draw deck. A player may choose to play any of the treasure or action cards by clicking on the card image. Action cards may only be played if the player has actions points available. If a card can be legally played, it is moved from the player's hand to the discard pile and the player may draw additional cards, gain coins to be used for the buy phase, or be given extra actions so they can play additional action cards. What the player gets to do depends on which action card they played. Treasure cards do not require actions and always increase the amount of buy points available to the player.

When a player no longer can, or wishes to play more action cards, they may choose to add cards from the supply to their deck. To buy a card, a player needs at least one buy action, their buy points need to be at least as great as the cost, and the quantity of the card in the supply must be greater than zero. If the player can buy a card they may click on the supply pile and move one copy to their discard pile. When a player can't, or no longer wishes to buy cards, they end their turn. All cards in the played cards area, or remaining in the player's hand are moved to the player's discard pile. The player shuffles their discard pile and draws five new cards and the next turn begins.

The game ends either when the last province is purchased or when any three other supply piles are empty. When the game ends, the total value of victory points in each player's deck should be calculated. The player with the highest number of victory points wins the game, if it's a tie, the player who went first loses.

In future iterations of the game there should be a scoreboard that players can view on the games website. This way players can keep track of their improvements, and compare their best scores with others. This could be done by ranking the average points per turn that a player earns. The scoreboard

should have the player's name, the score, and the ten kingdom cards used. The score data should be stored for permanence.

Some type of multiplayer capabilities should be added in the future as well. Multiplayer capabilities would significantly increase the scope of the project. The ideal way would be a networked system with all players at different computers. Another option is the "hot seat" method where players take turns sharing one computer. The main difficulty with the second method is that certain cards require decision making on another player's turn. This could make the game cumbersome, and in some situations basically unplayable.

Hopefully A.I will be in the first release of this game. Since network multiplayer won't be in the first release, and it would be the only other way to give users a smooth experience. It makes sense to at least have the option to play against the computer.

A Gamelog could be added to the game eventually. This is necessary for multiplayer, but may be helpful for single player as well. In a multiplayer game, it is important to be able to see what cards were purchased, played, or discarded on other players' turns. A solitaire player may wish to examine past games to improve their future play. If players have access to a gamelog, it's possible for them to critically examine games that they finished a long time ago. However, the potential audience for that is very limited and the gamelog should be added after multiplayer capabilities are implemented.

Adding additional cards in later updates would extend the lifetime of the game. So far 268 unique Kingdom Cards have been officially released through the base game and the expansions. An additional 33 are being released in October 2017 bringing the total up to 301. That means that there are more than 400 quadrillion unique combinations of 10 cards. In October, that number will rise to more than a quintillion. Each new card adds more variety than the last, making this an ideal target for future expansions.

This game should be released when the minimum requirements for an enjoyable experience have been met. Once the game is released and players are offering their feedback about the game, less important features can be added, and some features and game fixes that players request can also be added.

Glossary of Terms

Treasure cards consist of copper, silver, and gold cards. These cards are placed in the supply after each player takes 7 copper cards. These cards generate coins.

Victory cards Estate, Duchy, and Province cards are the basic victory cards, and are available in every game. If 3-4 people are playing, 12 (or 8 if 2 people are playing) each of Estate, Duchy, and Province cards are placed face-up in the supply after each player takes 3 estate cards.

Curse cards are mostly used with specific action cards like Witch cards. 10 curse cards go in the supply for a 2 player game, 20 cards go in the supply for 3 players, and 30 cards go in the supply for 4 players. These have a negative victory point value.

The **trash pile card** is a place where players put cards that were trashed in the game.

10 **Kingdom cards** are selected by the players at the beginning of the game. 10 of each of the selected cards are placed face-up in piles on the table.

Most Kingdom cards are **action cards** which have effects like generating coins, when a player is in their action phase.

Action phase is when the player can play an action card for every action they have, then follow the card's instructions.

Buy phase is when players can play any of their treasure cards that they want to, which will generate coins.

Clean-up phase is when players collect their hand and their cards and place them into their discard pile.

Actions refers to the number of action cards a player can play. Players start with 1 action, but sometimes cards can grant extras.

Buys: Players can increase their buys when they play treasure cards. Buys allow the player to purchase cards from their supply.

Coins are the main currency in dominion. Each card has a certain cost in coins. Playing treasure cards and some action cards will produce coins.

Cost is the amount of coins that must be paid to buy a card.

Deck: A set of cards. This can be a draw pile, discard pile, cards in hand, or a combination of those three

Discard: To move a card into the discard pile. This is done during a clean-up phase, or when another card has an effect that requires a card to be discarded.

Discard Pile: The pile that discarded cards should be placed in.

Draw: This is when a player takes a card from their deck and puts it in their hand.

Hand: A player's hand consists of the cards they can play.

Play Area: The table, or other surface where the entire game takes place. This is where all the decks and piles reside, and where the players gather around.

Supply refers to all the cards that players can buy in the game.

Supply Pile is the deck of cards that players can buy from.

Value: This refers to the amount of coins that must be used to buy the card.

Play Game: A button that when pressed, will start an instance of the game, or resume one that's currently in progress.

Play Card: When a player plays a card, they gain whatever actions or money the card is worth, and place the card in the trash pile.

Buy Card: When a player buys a card, they use the amount of coins that the card is worth, and add the card to either their hand, or their discard pile.

End Turn: Players push this button when they don't have any moves left. This initiates the next players turn.

System Requirements

Color display with minimum of 1024 X 768 resolution
1.5ghz CPU with integrated graphics
2GB of RAM
500MBs of free hard drive space
Mouse
Keyboard
Internet connection with minimum bandwidth of 56Kbps
Chrome, Firefox, or Edge Internet browser

Functional Requirements

Identifier	Priority	Requirement
Req1	10	The system shall initialize the board by selecting 10 random Kingdom cards, creating the supply with those cards, giving each player a shuffled starting deck and drawing five cards when a new game begins.
Req2	10	The system shall move the top card from the deck to a player's hand whenever a draw is required and the deck is not empty.
Req2.1	10	The system shall move the contents of the discard pile to the deck and randomize the card order when a draw is required and the deck is empty. The system shall then draw a card.
Req3	10	The system shall increase the number of coins available to a player by the value of the treasure card whenever a treasure card is played.
Req4	10	The system shall buy a card when a supply pile is clicked on by the player provided that player has at least one buy and at least as many coins as the cost of the card.
Req4.1	10	The system shall decrement the number of buys available when a card is bought.
Req4.2	10	The system shall decrease the coins available to the player by the cost of the bought card when a card is bought.
Req4.3	10	The system shall move one card from the appropriate supply pile to the purchasing player's discard pile when a card is bought or gained
Req4.4	10	The system shall check to see if the supply pile is empty when a card is bought or gained. If it is, the system will check to see if the game has ended.
Req4.5	10	The system shall set available actions to zero when a card is legally bought.
Req5	10	The system shall move all played cards and all cards remaining in the player's hand to the discard pile followed by drawing five new cards when a player indicates that their turn has ended.
Req6	9	The system shall end the game when there are three empty supply piles or when the Province supply pile empties.

Identifier	Priority	Requirement
Req 6.1	9	The system shall calculate each player's score by adding the value of the victory cards when the game has ended
Req 6.2	9	The system shall display an end game screen with the calculated scores of all players when the game ends.
Req7	8	The system should perform the text on an action card whenever an action card is played. For example, if a *Smithy* card is played, the player will draw 3 cards.
Req8	5	The system should allow a player to view a high score board sorted by the most efficient games played.
Req9	4	The system should give a player starting a new game the option of playing against a very simple AI opponent.
Req10	2	The system should allow two to four players to play over a network.
Req11	2	The system should record all changes to the game state and allow players to export it after the game is finished.

Nonfunctional Requirements

Identifier	Priority	Requirement
Req12	7	The system should allow players to play a card from their hand, buy a card, or end their turn with a single click
Req13	7	The system should take no more than ten seconds to initialize a new game
Req14	7	The system should take no more than three seconds to complete an action card
Req15	7	The system should take no more than five seconds to show the victory screen when the game has ended.

On-Screen Appearance Requirements

Identifier	Priority	Requirement
Req16	10	The system shall display all cards in their appropriate locations (the supply piles, the deck, the player's hand, the cards that have been played, and the discard pile)
Req17	9	The system shall display the current player's stats (actions, buys, buy points)
Req18	8	The system shall display the quantity of cards in the supply piles and the draw deck

Traceability Matrix

Reqt	PW	UC1 Play Game	UC2 Play Card	UC3 Buy Card	UC4 End Turn	UC5 Select Opponent	UC6 View High Scores	UC7 Export Log
R1	10	X						
R2	10	X						
R3	10	X	X					
R4	10	X		X				
R5	10	X			X			
R6	9	X						
R7	8	X	X					
R8	5						X	
R9	4					X		
R10	2					X		
R11	2							X
Max PW		10	10	10	10	4	5	2
Total PW		67	18	10	10	6	5	2

Above is the traceability matrix for this project. The play game use case is by far the highest priority. The next three highest are play card, buy card, and end turn. These are al included in or extend the play game use case. Play game and its subcases are where the bulk of our effort will be directed early in the project.

Effort Estimation (with use case points)

Use case	Description	Category	Weight
Play Game	6 steps for the main success scenario. 2 participating actors (player, opponents)	Average	10

Play Card	2 steps for the main success scenario. 1 participating actor (player)	Simple	5
Buy Card	2 steps for the main success scenario. 1 participating actor (player)	Simple	5
End Turn	4 steps for the main success scenario. 2 participating actors (player, opponents)	Average	10
Select Opponent Number	1 step for main success scenario. 1 participating actor (player)	Simple	5

Factor	Description	Weight	Score	TF
T1	Distributed system	2.0	0.5	1.0
T2	Response time/performance objectives	1.0	3.0	3.0
T3	End-user efficiency	1.0	1.0	1.0
T4	Internal processing complexity	1.0	0.6	0.6
T5	Code reusability	1.0	1.2	1.2
T6	Easy to install	0.5	0.0	0.0
T7	Easy to use	0.5	5.0	2.5
T8	Portability to other platforms	2.0	0.0	0.0
T9	System maintenance	1.0	1.0	1.0
T10	Concurrent/parallel processing	1.0	0.0	0.0
T11	Security features	1.0	0.5	0.5
T12	Access for third parties	1.0	0.0	0.0
T13	End user training	1.0	2.0	2.0
TOTAL				**12.8**

Factor	Description	Weight	Score	EF
E1	Familiarity with development process used	1.5	1.0	1.5
E2	Application experience	0.5	0.0	0.0
E3	Object-oriented experience of team	1.0	3.0	3.0
E4	Lead analyst capability	0.5	0.5	0.25
E5	Motivation of the team	1.0	2.5	2.5
E6	Stability of requirements	2.0	4.0	8.0
E7	Part-time staff	-1.0	5.0	-5.0
E8	Difficult programming language	-1.0	1.0	-1.0
TOTAL				6.25

UUCW = (3 * 5) + (2 * 10) + (0 * 15) = 35

UAW = (3 * 1) + (2 * 2) + (0 * 3) = 7

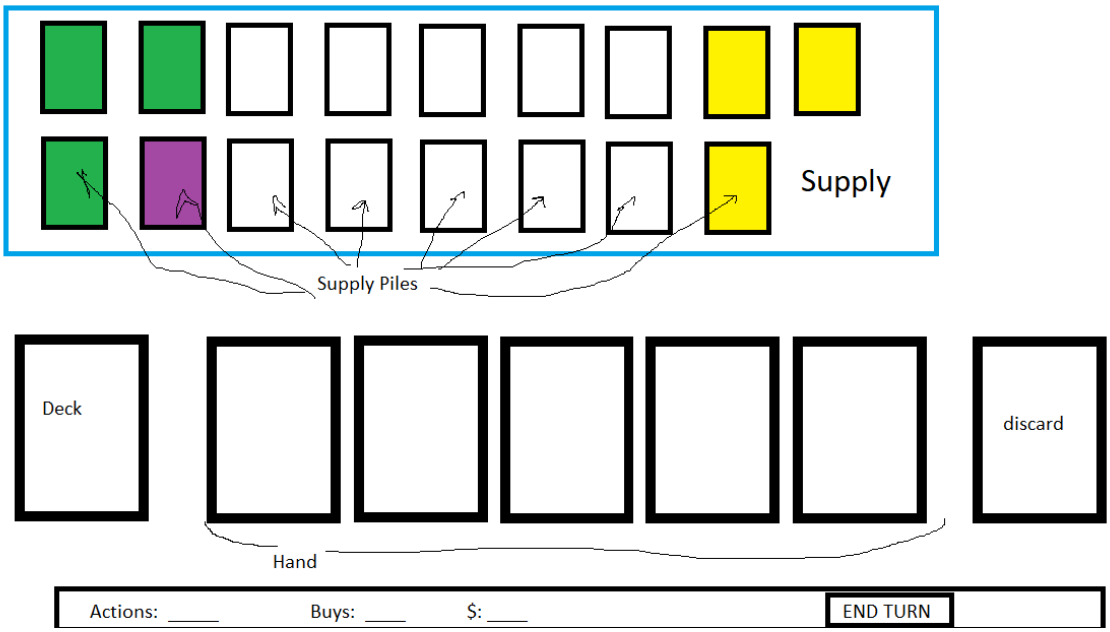
TCF = 0.6 + (12.8/100) = 0.728

ECF = 1.4 + (-0.03 * 6.25) = 1.213

UCP = (35 + 7) * 0.728 * 1.213 = 37.09

Estimated Effort = 37.09 * 28 = 1,038.52 Hours

Below is the layout for a potential UI:



Stakeholders

We have identified four major stakeholders:

S1: The customer/end user -- In this case, the customer and end user can be considered the same group. They are people who pay for and use this software.

S2: The design team -- These are the people in charge of making the necessary decisions to make this project a reality.

S3: Video game publisher – A publisher gets the completed game to the customers. It can either be an external publishing company or the design team could self-publish. Using an existing IP may limit the publishing options.

S4: Board game publisher – Dominion is an existing IP and the rights to digitally publish would have to be obtained from the publisher of the cardboard version. The publisher has been identified as Rio Grande Games.

Actors and Goals

This project has four actors:

A1: The Player – This is any person who uses our software to play a game of Dominion.

A2: Opponents – This actor only occurs in multiplayer vs human games. Opponents are special cases of the Player actor used when more than one player is affected by a given interaction. The initiating actor is considered the Player and Player actors are considered opponents. There may be multiple Opponents in a game, but they should be uniquely identified.

A3: High Score Board – This will be an external file or database that records the best games played.

A4: Game Log File – The system can record all the changes to a gamestate and a player may choose to export it to an external file.

Casual Use Cases

The summary use cases are as follows:

UC-1: Play Game – Allows the Player to play a game of Dominion.

Extension Point: Play Card – The player may play a card from their hand.

Extension Point: Buy Card – Allows the player to get new cards.

Inclusion Point: End Turn – Play passes to the next player.

Extension Point: Change Opponent – The Player may change the game type between solitaire, vs humans, and vs AI.

Derived from Reqs 1-7

UC-2: Play Card – Allows the player to play a card from their hand. The play must be legal and each card will give unique instructions that the system will follow.

(optional sub use case, «extend» UC-1: Play Game).

Derived from Reqs 3, 7

UC-3: Buy Card – Allows the Player to buy a card from a supply pile and add it to their discard.

(optional sub use case, «extend» UC-1: Play Game).

Derived from Req 4

UC-4: End Turn – A Player must indicate each time that their turn ends. The system will automatically finish any tasks required by the rules before the turn ends, and play will pass to the next player to begin their turn. In a solitaire game the player simply begins their next turn.

(mandatory sub use case, «include» from UC-1: Play Game).

Derived from Reqs 2, 5

UC-5: Select Opponent – Allows the Player to change from the default selection of a solitaire game to a multiplayer game against either human or AI opponents and back again. This use case requires a significant amount of resources and expertise that the development team does not currently have. Further research in this area is likely to add several currently unknown subcases, but given the low priority and limited time this use case is a placeholder until that research is completed.

(optional sub use case, «extend» UC-1: Play Game).

Derived from Reqs 9, 10

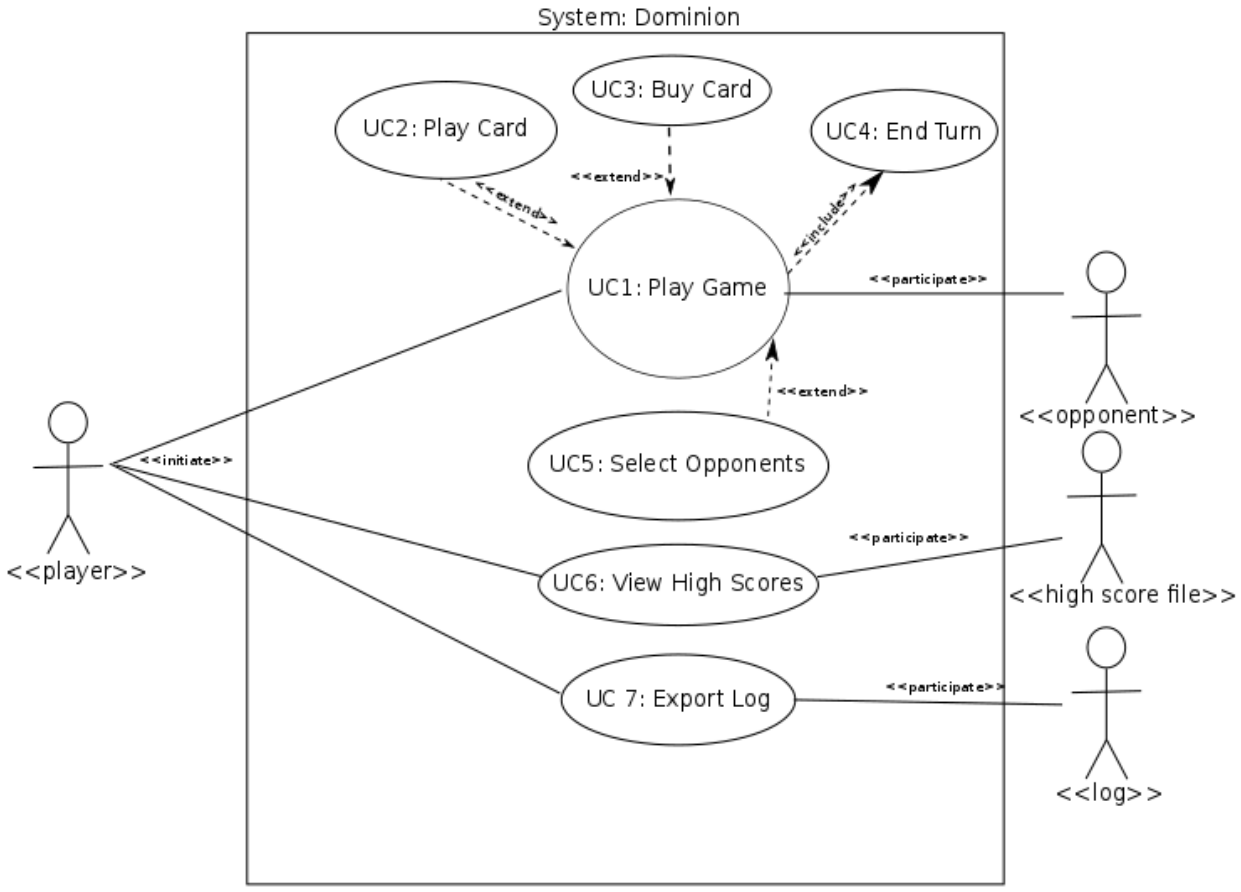
UC-6: View High Scores – Allows the player to see a list of the best games played. While there are many possible ways to define best, points/turns seems to be a reasonable metric of efficiency. The high score list may also include other information the player's name, the date, and the Kingdom Cards used.

Derived from Req 8

UC-7: Export Game Log – As the game is being played, the system should keep track of changes to the game state. When the game ends, the player may optionally export it to a file. Depending on the details of multiplayer implementation, the game log might be used by a player to see what happened on an opponent's turn.

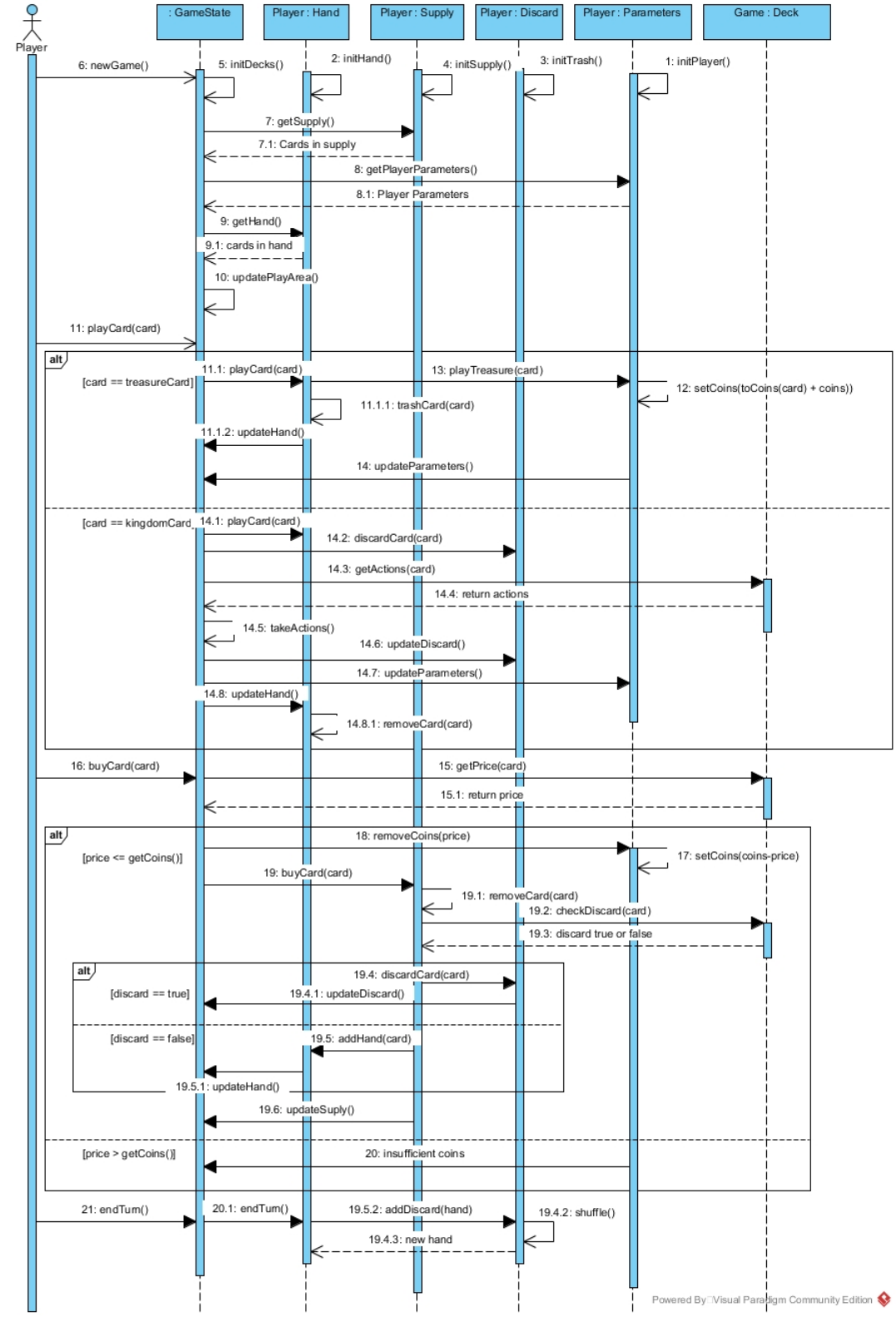
Derived from Req 11

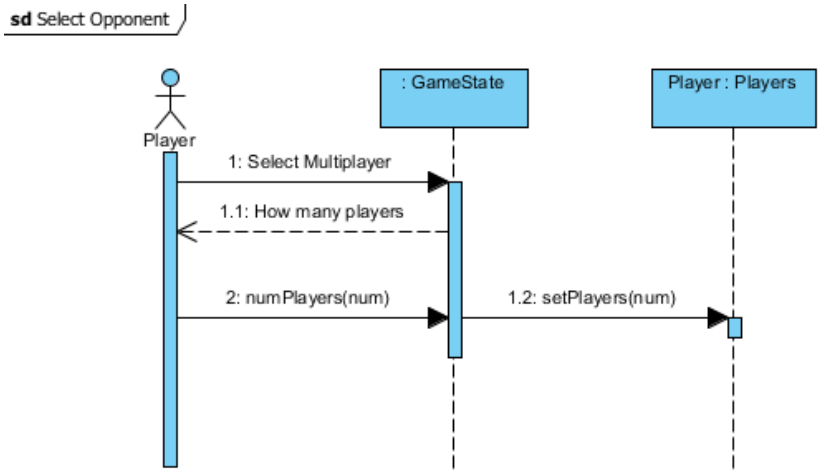
Use Case Diagram

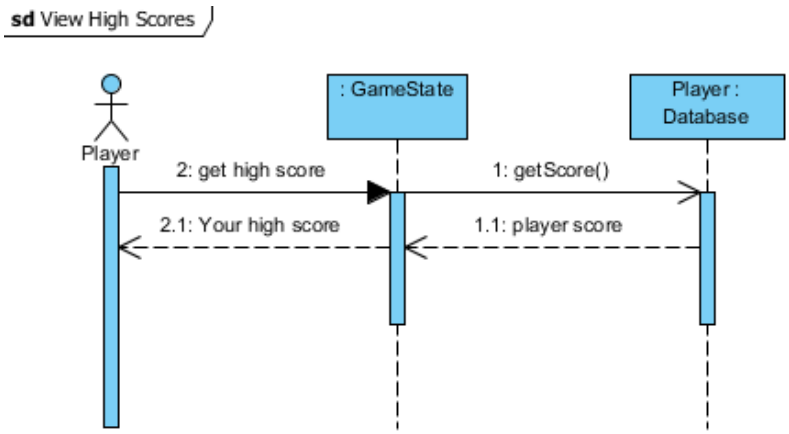


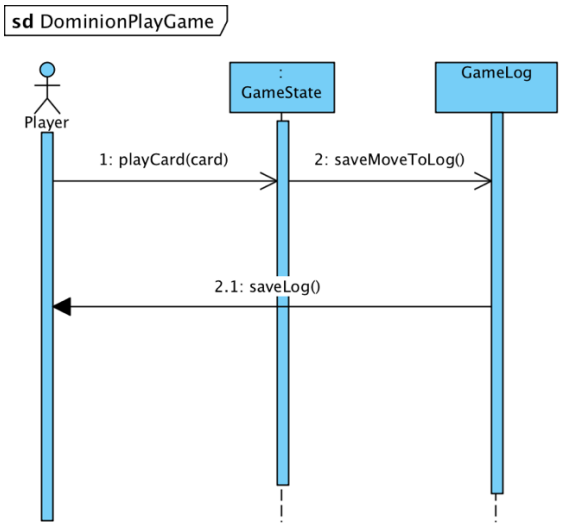
Interaction Diagrams

Sequence Diagram




sd Select Opponent
Player
: GameState
Player : Players
1: Select Multiplayer
1.1: How many players
2: numPlayers(num)
1.2: setPlayers(num)


sd View High Scores
Player
: GameState
Player : Database
2: get high score
1: getScore()
2.1: Your high score
1.1: player score

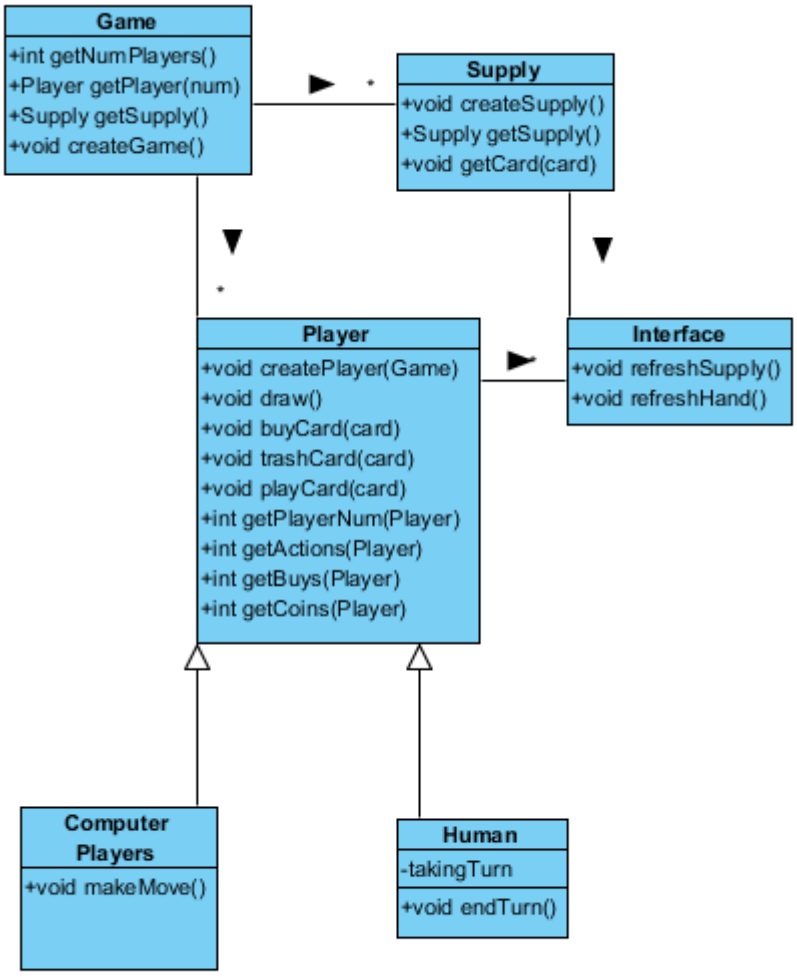

sd DominionPlayGame
Player
: GameState
GameLog
1: playCard(card)
2: saveMoveToLog()
2.1: saveLog()

Design Patterns

Chain of Responsibility – Using this design pattern we can pass requests two different objects when playing a card. This pattern will help make the prototype easier to read when playing a card.

Class Diagram and Interface Specification

Class Diagram



Data Types and Operation Signatures

Player Class

Data Type	Operation Signature	Description
Void	createPlayer(Game)	Creates a new player for the given game
Void	draw()	Draws one card from the current players deck and places it into their hand.
Void	buyCard(card)	Removes the specified card from the supply and places it into the players discard pile.
Void	trashCard(card)	Removes the specified card from a player's hand and places it into the trash pile.
Void	playCard(card)	Removes the specified card from a player's hand and places it in the trash or discard pile, and initiates the actions of the card.
Int	getPlayerNum(Player)	Returns the specified players number
Int	getActions(Player)	Returns the number of actions that the specified player has.
Int	getBuys(Player)	Returns the number of buys that the specified player has.
Int	getCoins(Player)	Returns the number of coins that the specified player has.

Game Class

Data Type	Operation Signature	Description
Int	getNumPlayers()	Returns the number of players currently in the game.
Player	getPlayer(num)	Returns the player with the specified number.
Supply	getSupply()	Returns the games supply deck
Void	createGame()	Creates a new game, and initializes all variables.

Supply Class

Data Type	Operation Signature	Description
Void	createSupply()	Initializes the supply for the game.
Supply	getSupply()	Returns the supply with all the cards.
Void	getCard(card)	Returns the object of the specified card.

Interface Class

Data Type	Operation Signature	Description
Void	refreshSupply()	Renews the supply displayed on the users screen
Void	refreshHand()	Renews the hand displayed on the users screen

Human Class

Data Type	Operation Signature	Description
Void	takingTurn	A state where the player is taking their turn
Void	endTurn()	Ends the player's turn and takes appropriate actions to end their turn.

Computer Players Class

Data Type	Operation Signature	Description
Void	makeMove()	Causes the AI to take their turn.

Design Patterns

The sequence diagram in this project uses a command design pattern. Objects in the program, which are functions in JavaScript, have their own methods encapsulated inside of them. When an object wants to call a method from a different object, it must go through that object first. The objects are game, supply, and player. For example, if a player wants to buy a card from the supply, the player object will call the supply object to initiate that action.

The class diagram for this project is a behavioral design pattern. One of the biggest problems to overcome while designing this game is to make sure entities can communicate with each other in a way that gives us enough flexibility to make the game work, but doesn't allow too much communication, which would result in an unstable program.

Object Constraint Language Contracts

Assertions:

- cardsPlayed.length == discardPile.length

Postconditions:

- After running function draw(), player.deck.length should equal 5.
- After running function endTurn(), these comparisons should be true..
  player.actions == 1;
  player.buys == 1;
  player.coins == 0;

Preconditions:

- if (player.deck.length == 0){ // This is the precondition
  if (player.deck.length == 0){
  player.deck = player.discardPile;
  player.discardPile = [];
  shuffle(player.deck);
  }
  }
- if (card.type === "coin") { // This is the precondition

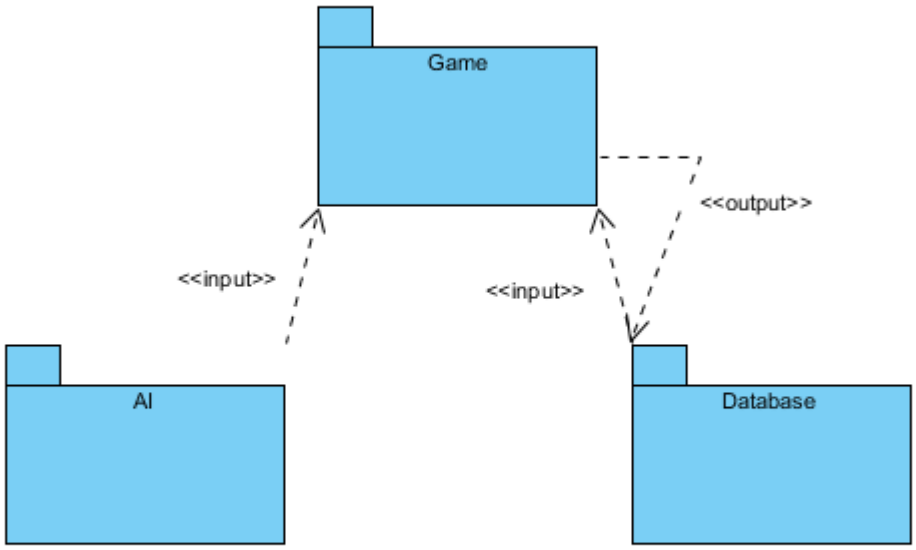
```
            player.coins += card.points;
            refreshStats(player);
            return true;
    }
```

System Architecture

Architecture Styles

The bulk of this system is built upon a **component-based** architectural style. The system is built from functions and objects that are dedicated to their own purpose so that they can be replaced, used in a similar program, and easily extended. This architectural style is easier to use than other architectures. In the future we may implement a **client-queue-client system** for multiplayer games.

UML Package Diagram



Mapping Subsystems to hardware

The subsystems are all client based for the time being.

Persistent Data Storage

The players high score needs to be persistent at some point in development. This may not be included in the initial release of the game. The player high scores will be stored in a relational database in the format:

Player name	High Score	Date

Network Protocol

We're using **HTTP** for this project because it's simple to use, and we want our game to be easily accessible so that users can simply go to our website and start playing the game.

User Interface Design and Implementation

From our initial mockups we ended up having to zoom-in the cards when the user hover them so they could read the content of the card. We added titles to the pile and trash to identify them without having to guess what they did. Moved the End Turn button, and the number of buys, actions and money above the hand so the user can always see this information. We moved New Game and High Score to the top of the screen where they are not inside of the game, and more as secondary tools.

We simplified the actions of the game so the user can play the game without reading extended set of rules. Clicking on an action card will perform the action right away, no second click necessary. Clicking on any card that required to be bought, will perform the action right away and take the money away. If user does not have the money, it will then inform them after clicking the card.

New game button will re-shuffle the deck instantly, and get the player into a new game in seconds. While high scores will show as an overlay on top of the way so the game is no disrupted if the user clicks on it.

We will be removing the AI integration and keeping the game simpler without any internet connection required. We will also need to strip some of the more complex card functionality to avoid running out of time and have more time to debug and fix issues.

Design of Tests

Test Cases

Test Case ID	Test Case	Expected Outcome
1	User clicks on card from supply to buy it, but has insufficient coins:	GetPrice() returns: "insufficient coins"
2	User clicks on card from supply to buy it and has sufficient coins:	GetPrice() returns: "sufficient coins"
3	Three decks in the supply have been depleted	endGame() returns the player scores
4	User clicks "End Turn" button	EndTurn() returns a player object, and
5	User clicks "New Game" button	Page is refreshed
6	User clicks "High Scores" button	High Scores page opens in browser
7	User clicks on the trash card	trashCard() returns the next card the user clicks on

Test Coverage

Since all the code hasn't been written yet, the coverage may not be completely accurate. We could test all the arrays for the player hand, player discard pile, and all the supply arrays with their objects, but those are mostly static, and testing the count for every card pile seems counterproductive.

A good estimate for the test coverage would be about 60 percent, we can certainly add more tests as we add more code, but some of the things that could be tested don't need to be tested.

Integration Testing Strategy

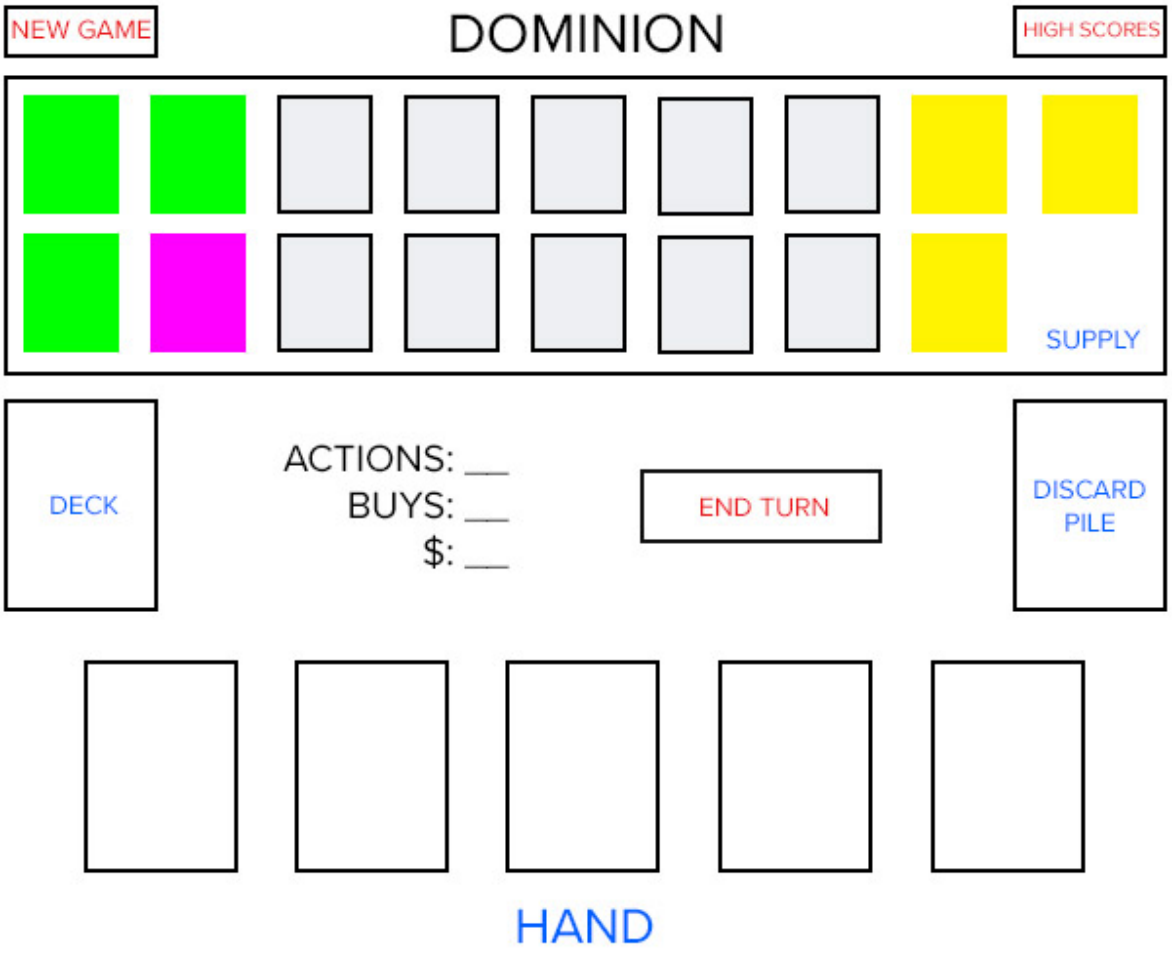
We plan to use a top down integration testing strategy so that we can start testing the program as a prototype. Top down also allows us to find major design flaws before smaller ones which is valuable if we can't complete as much as we'd hoped to.

History of work

So far, the plans for this program have stayed mostly the same. However, because we've lost a team member. Removing AI will save us a lot of time. Marked in red how we can device the rest of the work left before the second demo. All future work is marked with red.

	Brent	Carlos	Matt
Front-end Code	10%	90%	
Decks	80%		20%
Card Objects	95%		5%
Hand Deck	100%		
Discard Pile Deck		25% 75%	
Trash Deck		100%	
New Game Function	100%		
End Turn Function		100%	
Help tooltips	50%	50%	
A.I players	--	--	
Play Card Function	100%		
Debugging/Testing	25%	75%	

Preliminary Design



The game will start with the high score screen and the button to toggle New Game in the top corner. When the button is pressed the cards will show like the above mockup. The user will be able to click on any card in his hand to use the action. The deck is only for display and holding the cards, it will not display an active, same as the discard pile. Ending turn button will end the current turn. Actions, buys and $ have no button action, but will update dynamically. All the cards in the supply can be clicked on to buy with actions and/or buys.

If the “New Game” button is pressed anytime during a game, a warning sign will show if you would like to reset the game and start from zero.

High score button will activate the high score chart.

User Effort Estimation

NEW GAME

a) Click “New game”
b) Game starts

MOVE CARD TO TRASH PILE

a) Click “card” to move.
b) Click on “trash pile”.

USE ACTION/BUYS

a) If user has actions/buys, he can click on a card or buy a card.
b) Click on the card he wants to use, and move it to his hand.
c) Discarding the other card.

END TURN

a) Once all moves are done.
b) Click “End Turn”,
c) Next turn will be shown now.

SEE HIGH SCORES

a) Click “High Scores” button in the top right corner.
b) Click “X” (close button) once the user is done.

References

Vaccarino, D. X. (n.d.). *Dominion Game Rules.* Rio Grande Games.

Wikipedia. (n.d.). *Wikipedia*. Retrieved from Dominion (card game): https://en.wikipedia.org/wiki/Dominion_(card_gam