CSCI-466
CMP
Dominion Game
Bapar.5gbfree.com

October 17, 2017

Mathew Griffin
Carlos Perez
Brent Parker

Contributions:

Matthew Griffin:

Project management, and overlook of diagrams.

Carlos Perez:

Project management, Export Log diagram.

Family emergency

User interface design and implementation, progress report, integration and testing, merging contributions

Brent Parker:

Cover page, contents, select opponent and view high scores sequence diagrams.

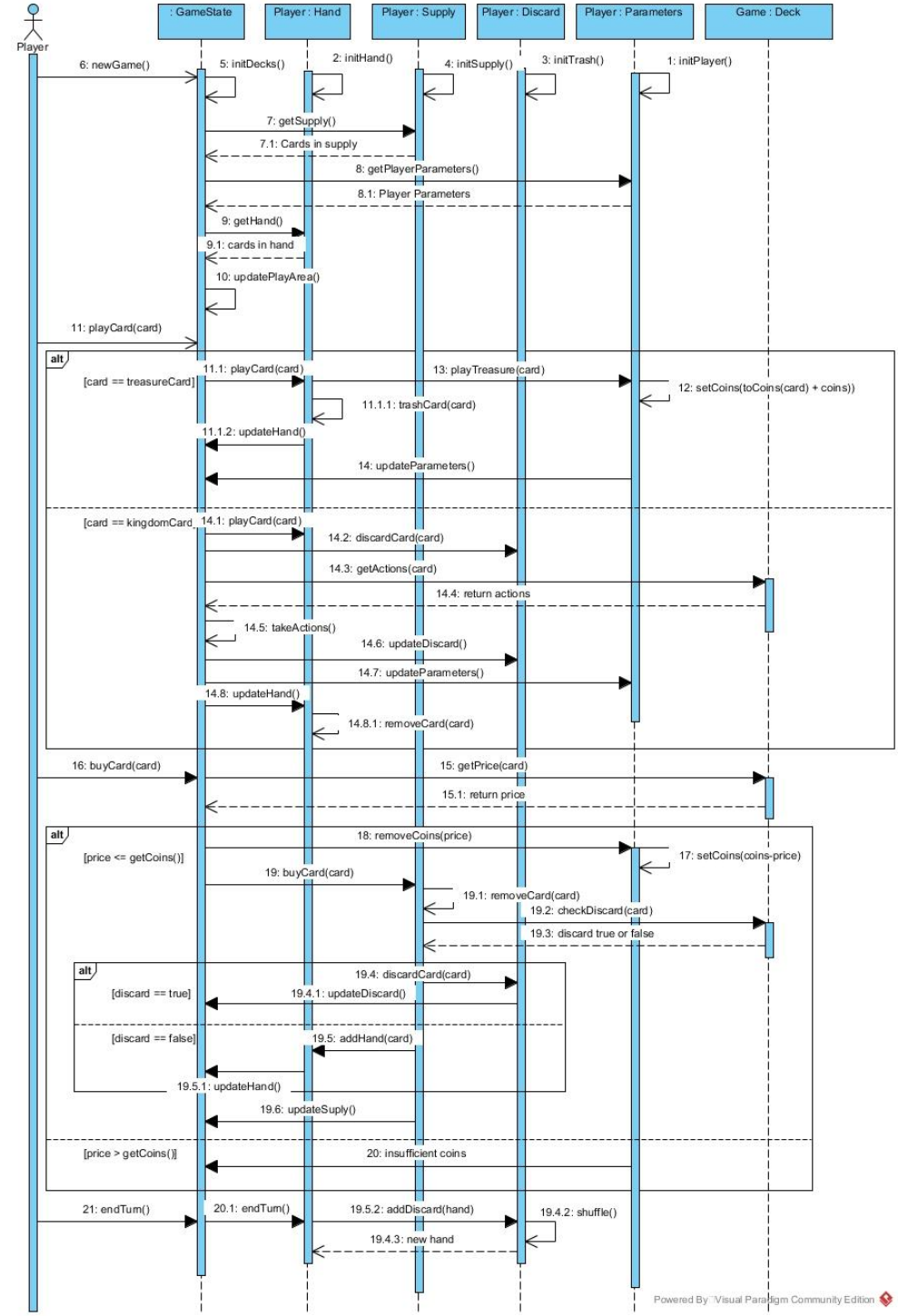
System Architecture, updated progress report.

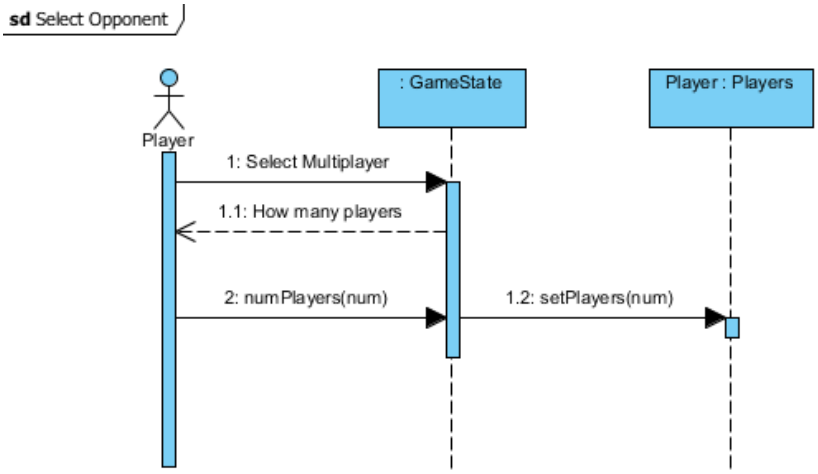
Data Structures, Design of Tests, class diagrams, data types and operation signatures.

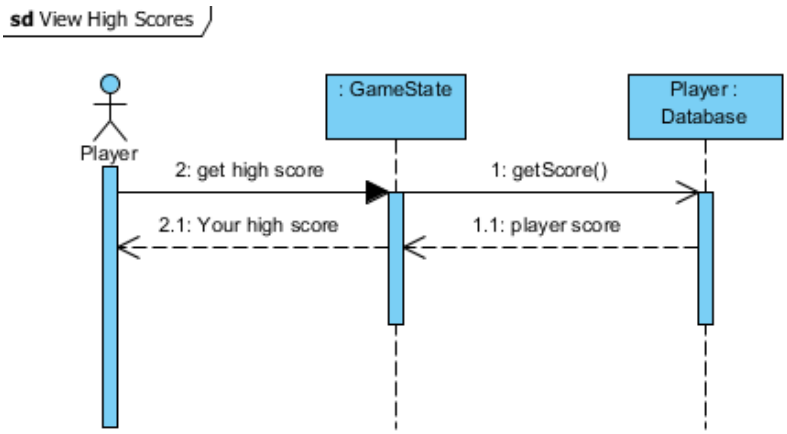
Contents

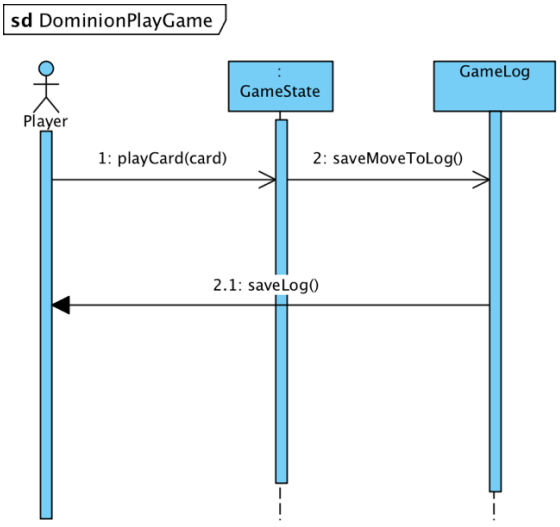
Interaction Diagrams

Sequence Diagrams



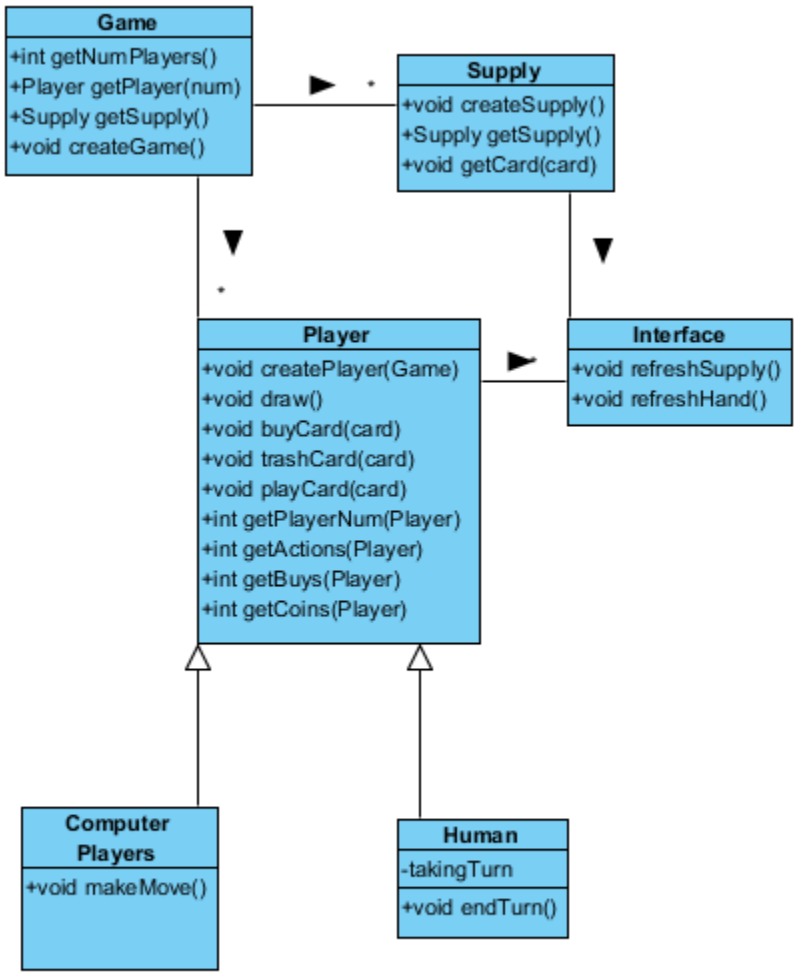

sd Select Opponent
Player
: GameState
Player : Players
1: Select Multiplayer
1.1: How many players
2: numPlayers(num)
1.2: setPlayers(num)


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()

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.

Data Type	Operation Signature	Description
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.

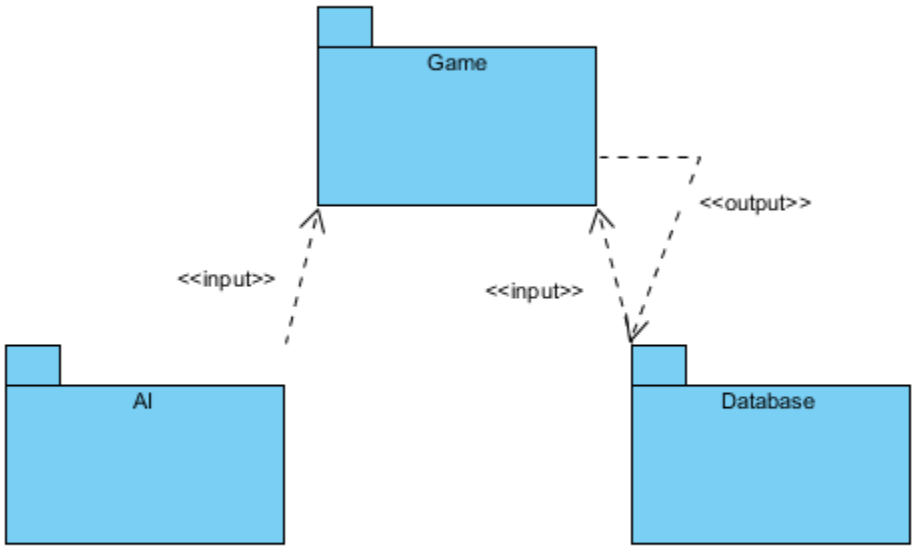
Traceability Matrix

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.

Global Control Flow

Execution Order:

The system will be event-driven. As with most websites the system will wait for the user to act before it does anything.

Time Dependency:

The system won't have any timers.

Hardware Requirements

- Color display with minimum of 1024 X 768 resolution,
- Minimum 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

Data Structures

We chose to use arrays to store all the cards in the form of objects for the game. Arrays made more sense than linked lists and trees because we know what size the array needs to be from the beginning, and we don't need to move elements around. We also need to access the elements out of order which also makes arrays more efficient for our purpose.

Arrays are less complex, use less storage, and are more efficient for the purpose we're using them for. We chose performance and flexibility.

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.

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.

Progress Report

	Brent	Carlos	Matt
Front-end Code		100%	
Deck	40%		50%
Make Objects of Cards	90%		10%
Hand Deck	25%		
Pile Deck		5%	
Trash Deck			
High Score			
New Game Function			
Help tooltips			
Network Functionality (optional)			

Integration and Testing

Integration is done as we go, the project all ties in together and different functions are coded while other members code others, so there is no need to integrate functions later on. Git will take care of merging our code.

We have two different weeks before each deadline (demo 1 and demo 2) dedicated to testing what we have so far. Testing has been assigned to all members of the team. This week will also act as catch up week if anyone is falling behind.

Merging Contributions

Code is merged through Github. All team members push their progress to git and all the code is stored and, once pushed other members can pull to test out the process. Since we're a small team we really don't use branches, we keep it simple.

Project Management

Week	Tasks
9/11 – 9/17	• Carlos: Design basic board the board, and place where elements will be placed. Design in CSS Kingdom cards and place in board. • Matthew: Design in CSS Treasure and Curse cards. Prepare all strings different cards will need. • Brent: Design in CSS Victory and Trash pile cards. For victory cards, generate the strings we will need • Goal for the week is to start to have our assets ready for functionality, and communicate with the team
9/18 – 9/24	• Finish task from last week and share with the team and get feedback from everyone. • Carlos: Functionality to start board's deck with 10 Kingdom cards and give the user 5 random cards from the deck. • Matthew: Treasure cards functionality. Assign the value of the cards, and basic functionality that will decrease the buy actions if clicked on a card. • Brent: Give the player their 7 copper cards and 3 estates cards (actions don't need to work perfect at this point, just make sure he gets the cards when he starts the game). • Goal for the week is to start with basic functionality that will be the framework of the game, and start shaping gameplay.
9/25 – 10/1	• Carlos: Action cards, once a player click on an action card. Perfect the action (coins, moves, whatever the card value/action says) • Matthew: Give a player a card after using his coins from the coin system. • Brent: Trash pile functionality. Move cards from the trash to the top supply, only if supply is empty. • Goal for the week is to get the card functionality in check, and perform quick moves with the cards.
10/2 – 10/8	• Carlos: When no more moves are available, and player wishes to move all cards to pile, also move 5 cards from supply into the deck • Matthew: Buy cards from the supply into a player's deck functionality and move card from supply to deck

Week	Tasks
	• Brent: End game logic, if there are no more cards to grab, or any 3 supply piles are empty, end the game and give final score/coins left. • This week's goal: Wrap up functionality as far as gameplay.
10/9 – 10/15	**Make up week**. If we are behind, or we need time catch up. Also test our code, and fix bugs from the functionality that should be complete.
10/16 – 10/22	• Carlos: Calculate final store, when a game ends. • Matthew: Treasure cards functionality. Assign the value of the cards, and basic functionality that will decrease the buy actions if clicked on a card. • Brent: Give the player their 7 copper cards and 3 estates cards (actions don't need to work perfect at this point, just make sure he gets the cards when he starts the game). • Goal for the week is to start with basic functionality that will be the framework of the game, and start shaping gameplay.
10/23 – 10/29	Prepare demo for next week. Use this time to fix final bugs, and present also prepare the report for the demo. Iron our bugs and prepare to turn in final demo one next week. Depending on the stage of the game, different people can pick up different bugs and functionality that needs to be done.
10/30 – 11/5	**DEMO 1 DUE 11/6** Make sure game is in good standing to turn in.
11/6 – 11/12	• Carlos: High score functionality, add up the score of the player and store. • Matthew: Rules of the game, add a tutorial to let the user know how to play the game. • Brent: Hint system. Give player tooltips for help, and add (?) icons for players to seek help if needed. • Work on UI, make the experience easier, and show the player how to work.
11/13 – 11/19	• Carlos: Work on animations, make the game more presentable. • Matthew: Iron out cards, graphics and work on CSS to make the game easier on the eyes.

Week	Tasks
	• Brent: CSS fixes, bugs, and make sure the game works on different browsers (chrome,ie,firefox).
11/20 – 11/26	**SHORT WEEK – HAPPY THANKSGIVING** **Easy week, fix bugs that we find and work on smaller items as we feel, but mostly taking it easy for this week.**
11/27 – 12/3	• Carlos: Fixes, bugs and documentation for next week. • Matthew: Documentation, reporting and final stages of the game. • Brent: Testing the game, provide feedback to others and work on a final bug list we can all work on during the week.
12/4 – 12/10	**DEMO 2 DUE 12/11 – FINAL VERSION** **Go over the list of final bugs and fixes that Brent worked on last week.**
12/11	**Turn in final Project**

References

DominionStrategy Wiki (n.d.) *DominionStrategy Wiki.* Retrivied from: http://wiki.dominionstrategy.com/

Guru99. (n.d.). *INTEGRATION Testing Tutorial: Big Bang, Top Down & Bottom Up*. Retrieved from Guru99: https://www.guru99.com/integration-testing.html

Microsoft. (n.d.). *Architectural Patterns and Styles*. Retrieved from microsoft.com: https://msdn.microsoft.com/en-us/library/ee658117.aspx?f=255&MSPPError=-2147217396