Design Document

Team PI-b

17 March 2019

Name	ID Number
Simon Huang	27067380
Jonathan Massabni	26337430
Alexia Soucy	40014822
Anthony Funiciello	40054110
David Gray	40055149
Mair Elbaz	40004558
Rani Rafid	26975852

TABLE OF CONTENTS

Contents

LIST OF FIGURES

List of Figures

1 INTRODUCTION

The purpose of this project is to use software engineering practices to develop a desktop application which plays the Codenames game with NPC's of varying ”intelligence.” The software was designed using MVC architecture and several common design patterns including the ”Strategy Pattern,” ”Observer Pattern,” and ”Command Pattern.”

This document provides an explanation of our application's architectural and class level software design. The Architectural Design section gives a high level look at how the system is organized using the Model-View-Controller (MVC) architecture. The Detailed Design section describes the design within the subsystems and the purposes of each class. The Dynamic Design Scenarios section explains how the application works by explaining significant execution scenarios of the system.

2 ARCHITECTURAL DESIGN

2.1 Architectural Diagram

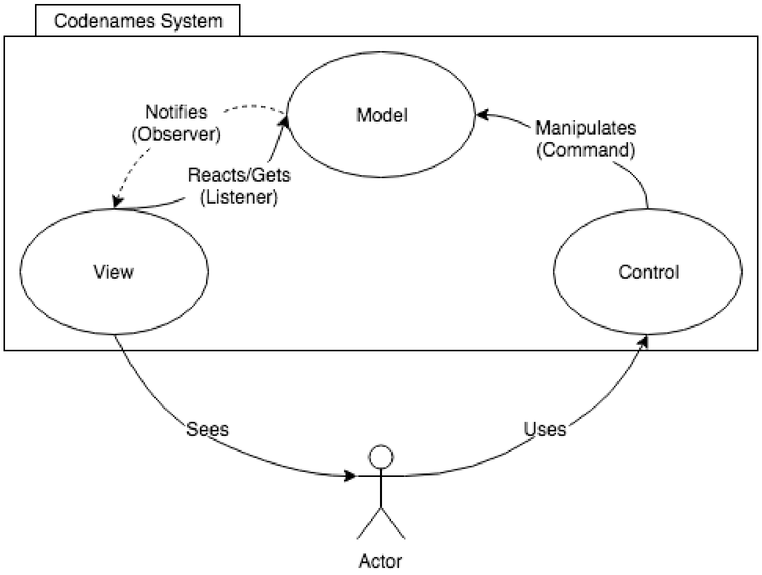

Figure 1: System Architecture based on Model-View-Control design

The Codenames game project specifications call for a graphical interface that can reflect an evolving game state and a system through which user input is processed to alter the game state. The application is a layered logical architecture composed of three subsystems inspired and named after the Model-View-Controller (MVC) separation principle. This principle provides a logical approach to dividing the domain, user interface, and application layers into subsystems that can be developed independently and, through interfaces, interact with each other. This section provides a description of the responsibility of each subsystem and discusses the reasoning behind choosing to model this systems architecture according to MVC.

The Model subsystem represents the current game state through its three component packages: Board, which represents the data relating to the ongoing game, Player, which represents the players and their strategies (using the Strategy pattern), and Util, which logs everything that takes place in Model. In general, the Model subsystem is synonymous to the domain layer, and is therefore an inspirational creation of the domain model.

The View subsystem contains all objects that provide a user-friendly interface to the Codenames game. These objects are responsible for capturing input, and output. They may sometimes produce graphical elements. However, they are not responsible for containing data pertaining to the state of the game, nor do they incorporate any logical behaviour affecting the internals of the application. In the Codenames application, the View subsystem uses the Observer pattern to bind itself to specific Subject classes in the Model subsystem to display the current game state in real time. This allows the Model subsystem to function uninterrupted, independently of the View subsystem.

The Control subsystem contains all objects tasked with defining the stimulus and/or handling stimuli forwarded by the View subsystem. In this context, a stimulus is defined as a request, or message that initiates work to be performed by the system. Furthermore, objects in this subsystem possess functional characteristics. In the Codenames application, the game progresses through one turn when the user presses the ”Enter” key. The action of a user pressing Enter is handled by the GameHandler, which through the Command design pattern, commands the Model classes to do the next turn, changing the Models state, and updating the View accordingly.

2.2 Subsystem Interface Specifications

In general, the Model and View according to the Observer design pattern, and the Controller controls the Model by making commands through the Command design pattern. This section specifies the actual function calls which make up these software interfaces between the three (MVC) subsystems. Outside of the two design patterns defined about, most of the Model's interface is functions called by View classes to get the internal state of the game, on which the GUI is based.

2.2.1 Model Subsystem Interface

The communication between the Model and View subsystems is implemented in the Observer Design Pattern. As a result, the Model contains classes which extend the Subject abstract class, to which Observer classes in the View may attach() themselves, so that they are notified to changes in the Subject's state. The GameManager, Verbose, and Card classes are Subjects.

Subject An abstract class for Subject's in the Observer design pattern.

- void attach(Observer o): add the Observer object o to the list of Observers to be notified upon changes of state.
- String getStringProperty(): returns the Subject's state in the form of a String.
- CardType getTypeProperty(): returns the Subject's state in the form of a CardType (Red, Blue, Assassin, Bystander).

GameManager The GameManager class keeps track of much of the state of the game. It is a Subject, serving as an interface for the View to get data on the games state. Along with the methods of Subject, the GameManager class has the following methods:

- void doNextTurn(): The main interface between the Controller and the Model. This initiates the Model to do the next turn, changing the state of the game.

- Clue getCurrentClue(): Returns the Clue given by the current team.

- int getBlueScore(): Returns the number of cards left for Blue to guess.

- int getRedScore(): Returns the number of cards left for Red to guess.

- CardType getWinner(): Returns void if nobody has yet won, or CardType.Blue, or CardType.Red if there has been a winner.

CardBuilder The CardBuilder class is a helper class for extracting the word data from the .txt files, and turning it into a randomized board. It is used to start a game.

- Card[] buildAll(): Returns 25 Cards, generated from the database of nouns and keycards. Used to initialize the game.

Verbose The Verbose class is used for logging in the Model. It is a Subject. It uses the Subject's getStringProperty() method to notify its Observers of log messages. It is a Singleton, and as a result has the following methods:

- get() Returns the one instance of Verbose.

- bind(Observer) attaches Observers to the instance of Verbose.

2.2.2 View Subsystem

The View subsystem will start by calling start(Stage) that will create a new verboseView and Verbose object. Codenames class is also responsible for building the GameScene this class calls build(Subject[], Event<KeyEvent>) of type scene that will set up user input to send to the CardPane class. After GameScene create a CardPane object, it will hold the images of the cards and where each card is place on the board to display. The Verbose object will call bind(Observer) to bind an Observer object to the game board to record events. log(String) will capture events and convert them to phrases that will display in VerboseView. Lastly, a Subject object can call attach(Observer) to attach a new Observer to it which will keep track of things in the Model subsystem.

CardPane A class that implements the Observer to represent the GUI of 1 card on the board. It is covered until the update function is called.

- update() Called by subject of class to reveal color of card with image.

2.2.3 Control Subsystem

The control subsystem creates the command package. Codename's start(Stage) will create a new GameHandler object. This will contain information about the model's GameManager, the View's score and verbose and the control's commandManager and store a history of used commands. The handle(KeyEvent) will control the action on which key is pressed to execute the appropriate command and change the state of the game in the model. After GameHandler, creates a CommandManager, it will be able to create and execute() Command objects. GameHandler also makes a NextTurnCommand object that controls the turn flow of the game to set the input to match the game's commands.

The Difficulty class sets the different strategies that each player type will be able to use. Using setDifficulty(), the user is able to select their difficulty and use the appropriate strategy that is stored in the model subsystem.

GameControls A class responsible for handling the events of restarting or quitting the game. The game board will be reset upon restarting the game or will terminate upon selecting quit.

- setEvents() Sets the menu items and actions for the game which includes (Restart) and (Quit).
- setAbout() Sets the description of the menu items in a new window.

3 DETAILED DESIGN

The Codenames class is the entry point of the program. It instantiates classes from the Model, Control, and View subsystems, and connects them together. Because it starts and links major parts of the system, it exists outside of the other major subsystems.

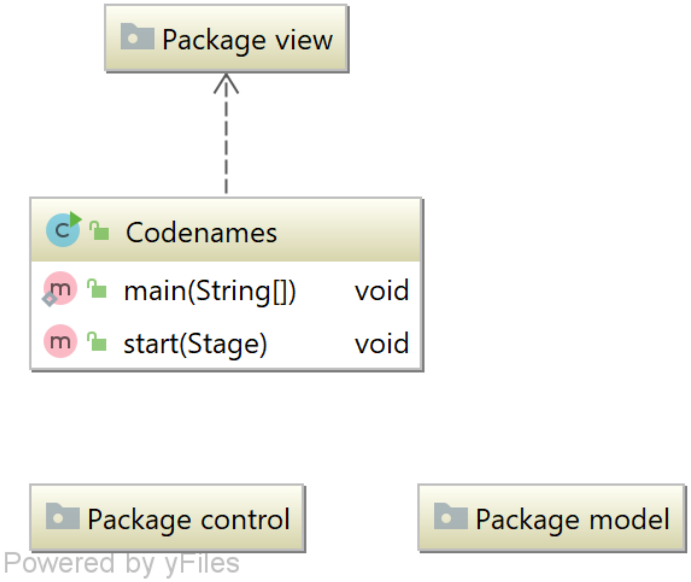

Figure 2: UML Diagram of the System Codenames

Class Name	Codenames		
Inherits From	Application		
Description	Main entry point of the program. Engine for running the Codenames game.		
Methods	Visibility	Method Name	Description
	Public	main(String[] args)	Launches the application's routine, including start method
	Public	start(Stage root)	Initializes the cards, board, and viewers. Binds the viewers to the cards and board. Shows the main game scene.

3.1 Subsystem Model

The Model subsystem contains the business logic of the system. The Model contains classes which represents the game data and state. It is organized into a player package, a board package, and a util package. The board package contains the classes representing the Codenames game being played, modeling the 25 codename cards and the keycard. The player package contains classes modeling the simulated players and their different play styles. The util package contains the Models logger.

3.1.1 Detailed Design Diagram

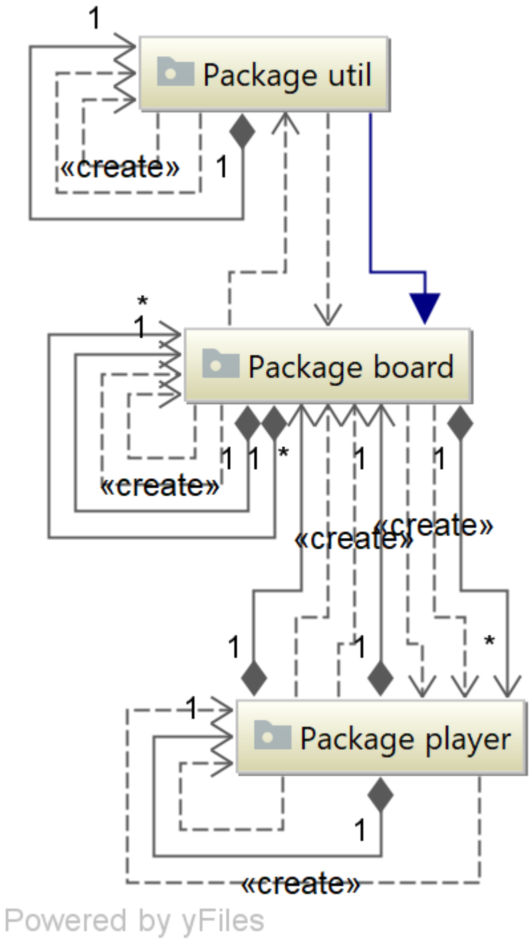

Figure 3: UML Diagram of Subsystem Model

3.1.2 Package Board

The Board subsystem of the Model contains the classes which represent the data and state of the Codenames game. This includes classes for game objects such as Card, Board, KeyCard, Clue, and CardType. The subsystem also contains classes which are used to initialize these game objects. The GameManager class includes references to the game objects being played with, and the Players playing the game. The GameManager also contains methods which modify the game state by making players take turns. The Controller's Commands to the Model contain function calls to the GameManager class.

3.1.2.1 Detailed Design Diagram

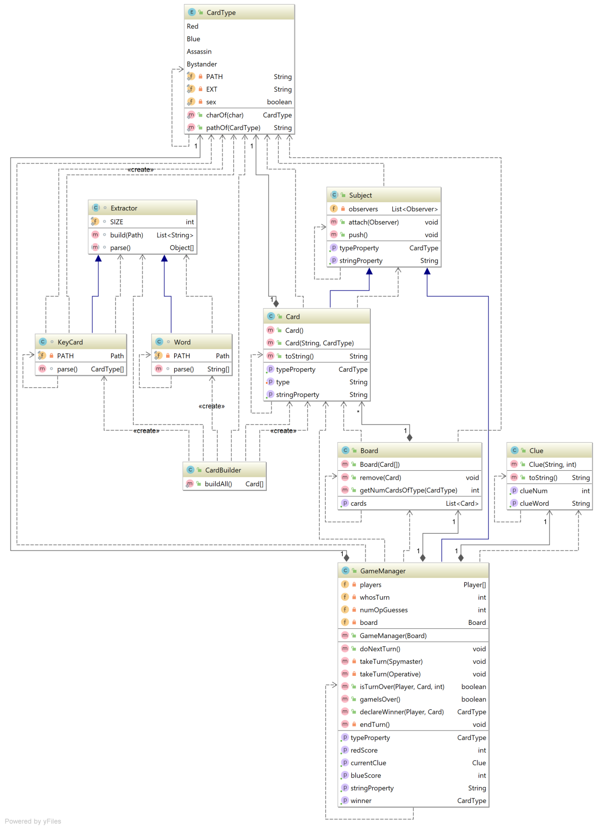

Figure 4: UML Diagram of Package Board in Module Model

3.1.2.2 Class Bipartite

Class Name	Bipartite
Inherits From	None
Description	Creates a bipartite graph showing the relation between words and clues.

Attributes	Visibility	Data type	Name	Description
	Private	HashMultiMap ⟨String, String⟩	wordsToClues	The map from words to clues.
	Private	HashMultiMap ⟨String, String⟩	cluesToWords	The map from clues to words. (One clue can be associated with multiple words)
Methods	Visibility	Method Name		Description
	Public	Bipartite(Board board)		Constructor; creates empty hashMultiMaps and sets them to the board's variables, then processes them.
	Private	processCards(Board board)		Creates a bipartite graph with only the words on the current board.
	Public	getWordsToClues()		Returns the hash map that contains the mapping of words to clues.
	Public	getCluesToWords()		Returns the hash map that contains the mapping of clues to words.
	Public	getClue(String cardWord)		Returns a string representation of a clue.
	Public	removeWord(String word)		Removes a word from the bipartite including its related clues.
	Private	debug()		Prints the current status of the bipartite.

3.1.2.3 Class Board

Class Name	Board			
Inherits From	None			
Description	The board stores the set of code name Cards on the board, which have not yet been guessed.			
Attributes	Visibility	Data type	Name	Description
	Private	List of Cards	cards	Holds the set of Cards that have not yet been guessed by Operatives
Methods	Visibility	Method Name		Description

	Public	Board(Card[] cards)	Initializes the board with a list of cards.
	Public	remove(Card c)	Remove the specified card from the set of unchosen cards.
	Public	getCards()	Return all of the unchosen cards still available, as a List.
	Public	getNumCardsOfType (CardType type)	Return integer count of the number of unchosen cards left with color specified by variable type.

3.1.2.4 Class Card

Class Name	Card			
Inherits From	Extends Subject			
Description	The Card class represents a single codenames card, and it's true identity (color). It is a subject in the observer pattern, because the view's CardPanes each individually observe a card.			
Attributes	Visibility	Data type	Name	Description
	Private	String	word	The codename word.
	Private	CardType	type	The true identity of the card: Blue, Red, Assassin, or Bystander.
Methods	Visibility	Method Name		Description
	Public	Card()		Default constructor; initializes word and type to null.
	Public	Card (String word, CardType type)		Constructor.
	Public	setType(String word)		Sets the codename on this card.
	Public	toString()		Returns the codename. Could be changed to return color and word.
	Public	getStringProperty()		Returns the codename. This overrides a method of Subject.
	Public	getTypeProperty()		Returns the cards true identity (color). This overrides a method of Subject.

3.1.2.5 Class Extractor

Class Name	Extractor			
Inherits From	None			
Description	Abstract class extractor abstracts the process of ingesting 25 random lines of a file. This process is done in creating the KeyCard from the database of keycards, and for choosing 25 codenames.			
Attributes	Visibility	Data type	Name	Description
	Private	final int	SIZE	The number of cards in a board. 25.
Methods	Visibility	Method Name		Description
	Public	build(Path path)		Returns every line of the file at Path path as a list.
	abstract	parse()		To be overridden by any Extractor, to return a list of whatever objects the class is creating from the data it extracts.

3.1.2.6 Class Word

Class Name	Word			
Inherits From	extends Extractor			
Description	Extracts words from a file and returns the first 25.			
Attributes	Visibility	Data type	Name	Description
	Private	final Path	PATH	The path to the .txt file containing the codenames.
Methods	Visibility	Method Name		Description
	Public	parse()		Calls build() and returns only the first 25 elements of the randomly ordered list of Strings (to be codenames).

3.1.2.7 Class KeyCard

Class Name	KeyCard		
Inherits From	extends Extractor		
Description	Extracts all of the data representing KeyCards from a text file, then chooses a random one and parses it into a List of CardTypes.		
Methods	Visibility	Method Name	Description
	Public	parse()	Uses build() to get the possible Key Cards in a random order, then takes the first one and maps each character to a CardType to create a list of CardTypes.

3.1.2.8 Class CardBuilder

Class Name	CardBuilder		
Inherits From	None		
Description	Creates an array of Card objects based on a random selection of 25 words and a key card.		
Methods	Visibility	Method Name	Description
	Public	buildAll()	Use the KeyCard and Word classes to create an array of 25 Cards (the core of the board).

3.1.2.9 Class CardType

Class Name	CardType			
Inherits From	None			
Description	An enum type which represents the possible colours of a card on a Key Card. Blue, Red, Assassin, or Bystander. Can also be used for team colour.			
Attributes	Visibility	Data type	Name	Description
	Private	String	PATH	The path to the images used by the GUI to represent the colours on the board.
	Private	String	EXT	The file extension of the images used by the GUI to represent the colours on the board.
	Private	boolean	sex	For the GUI, some cards should be displayed as male spys and some should be female.
Methods	Visibility	Method Name		Description
	Public	charOf(char arg)		Each CardType is represented in the database as a character in a String. charOf returns the CardType represented by a character. B is Blue, R is Red, Y is Bystander, A is Assassin.
	Public	pathOf(CardType type)		Returns the path to the image which can be used to display this colour on the board GUI.

3.1.2.10 Class Clue

Class Name	Clue

Inherits From	None			
Description	Represents a Clue given by a SpyMaster. Keeps track of the clue word and the number of associated cards.			
Attributes	Visibility	Data type	Name	Description
	Private	String	clueWord	The word part of the clue.
	Private	int	clueNum	The number part of the clue. Should represent the number of Cards associated with the word.
Methods	Visibility	Method Name		Description
	Public	Clue(String clueWord, int clueNum)		Constructor.
	Public	getClueWord()		Getter for the clue word.
	Public	getClueNum()		Getter for the clue number.
	Public	toString()		Clue represented as a string. Could be ”Word:Num”.

3.1.2.11 Class Constants

Class Name	Constants			
Inherits From	None			
Description	Stores any constants used across classes			
Attributes	Visibility	Data type	Name	Description
	Public	Path	WORDS_PATH	Path to the words file; used for debug.
	Public	boolean	DEBUG	Debug mode. True.

3.1.2.12 Class Subject

Class Name	Subject			
Inherits From	None			
Description	Part of the Observer pattern. Classes which are to be observable (the subjects) extend this class.			
Attributes	Visibility	Data type	Name	Description
	Private	List of Observers	observers	The set of Observers observing this subject.
Methods	Visibility	Method Name		Description
	Public	attach(Observer observer)		Adds observer to this Subject's list of observers.

	Public	push()	Notify all observerrs that the state of this Subject has changed.
	Public	getStringProperty()	Abstract method to be overridden by all Subjects. This is a way for Observers to get state information from the Subject, as a String.
	Public	getTypeProperty()	Abstract method to be overridden by all Subjects. This is a way for Observers to get state information from the Subject in the form of a CardType (a color).

3.1.2.13 Class GameManager

Class Name	GameManager			
Inherits From	None			
Description	Keeps track of the games state, including the players, current turn, and current clue. Allows manipulation of game state by initiating the next turn.			
Attributes	Visibility	Data type	Name	Description
	Private	Player[]	players	Array of players participating in the game. Should be 4 players, in order of when they will take their turn. So Red Spy, Red Op, Blue Spy, Blue Op.
	Private	int	whosTurn	Index into players to keep track of whos turn it currently is.
	Private	CardType	winningTeam	Holds the colour of the winning team. Not set if nobody has won yet.
	Private	Clue	currentClue	Reference to the clue given by the team who is currently taking their turn.

	Private	int	numOpGuesses	Keeps track of how many guesses the operative has made so far in their turn, as the game must enforce a limit.
	Private	Board	board	Reference to the Board object the game is being played on.
	Private	Bipartite	bipartite	Shows the relationship between words and clues.
Methods	Visibility	Method Name		Description
	Public	GameManager(Board board)		Constructor. Start a game with the board.
	Public	doNextTurn()		Make the game run the next turn, changing the state of the game.
	Private	takeTurn(Spymaster p)		Used by doNextTurn(), if the current player is a Spymaster. Makes the Spymaster take its turn.
	Private	takeTurn(Operative p)		Used by doNextTurn(), if the current player is an Operative. Makes the Operative take its turn.
	Public	isTurnOver(Player p, Card guess, int clueNum)		Tells whether or not the current turn should end based on current state information given.
	Public	gameIsOver()		Returns true if a winning team has been determined.
	Public	declareWinner(Player lastPlayer, Card lastGuess)		Determines the winner based on the last player's team and what card they last guessed.
	Public	endTurn()		Ends the turn and resets the number of guesses.
	Public	getBlueScore()		Returns the blue team's current score.
	Public	getRedScore()		Returns the red team's current score.
	Public	getWinner()		Returns the winning team.

	Public	getCurrentClue()	Returns the current given clue.
	Public	getStringProperty()	Returns either the current clue or a game over message, depending on current game state.
	Public	getTypeProperty()	Returns either the team whose turn it is, or the winning team, depending on game state.

3.1.2.14 Class JSONProcessor

Class Name	JSONProcessor		
Inherits From	None		
Description	Processes a .json file and return a JSON object. Using Simple JSON library		
Methods	Visibility	Method Name	Description
	Public	ProcessCurrentJSON()	Processes the current .json file defined in Constants.
	Public	ProcessCustomJSON(Path path)	Process any kind of .json file
	Public	ProcessJSON(Path path)	Process .json object. If the specified file does not exist, the program will terminate and display the error

3.1.3 Package Player

The Player subsystem of the Model contains the classes which represent Players of the Codenames game. The two types of players, Spymaster and Operative are subclasses of the Player class. Spymasters, and Operatives are able to have different play strategies, implemented using the Strategy design pattern. The classes of the Model which model the Players and their play styles are all a part of the Player subsystem.

3.1.3.1 Detailed Design Diagram

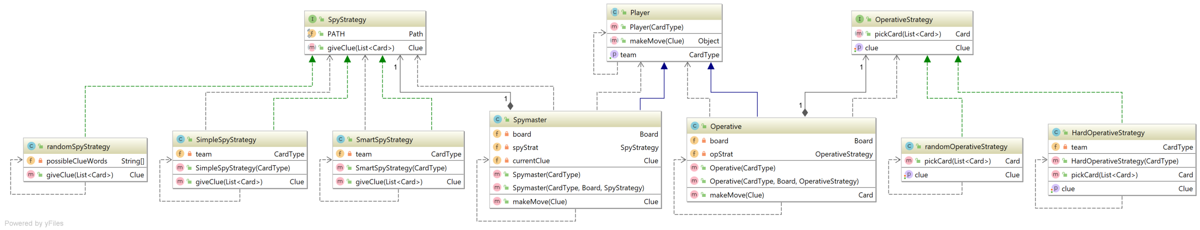

Figure 5: UML Diagram of Package Player in Module Model

3.1.3.2 Class Player

Class Name	Player			
Inherits From	None			
Description	The abstract class for all players. (Spymasters and Operatives)			
Attributes	Visibility	Data type	Name	Description
	Private	CardType	team	The player's team (red or blue).
Methods	Visibility	Method Name		Description
	Public	Player(CardType team)		Constructor. Sets the player's team.
	Public	getTeam()		Returns the player's team.
	Public	makeMove(Clue clue, Bipartite bipartite)		Implemented by Operatives and Spymasters. Operatives return a card, Spymasters return a clue.

3.1.3.3 Class Operative

Class Name	Operative			
Inherits From	extends Player			
Description	The implemented class for operatives. Defines the basic functions and constructor that the operative strategies use.			
Attributes	Visibility	Data type	Name	Description
	Private	Board	board	The player's board they will play on.

	Private	OperativeStrategy	opStrat	The strategy the player will use.
Methods	Visibility	Method Name		Description
	Public	Operative(CardType team)		Constructor. Instantiates the operative's team.
	Public	Operative(CardType team, Board board, OperativeStrategy strategy)		Constructor. Instantiates the operative's team and sets their board and strategy.
	Public	makeMove(Clue clue, Bipartite bipartite)		Picks a card based on the operative strategy in use.

3.1.3.4 Class OperativeStrategy

Class Name	OperativeStrategy		
Inherits From	None		
Description	The interface for operative strategies. (Strategy pattern)		
Methods	Visibility	Method Name	Description
	Public	pickCard (List⟨Card⟩ cards, Bipartite bipartite)	Operatives pick a card from a list of cards.

3.1.3.5 Class BotOperativeStrategy

Class Name	BotOperativeStrategy			
Inherits From	implements OperativeStrategy			
Description	Operative strategy used by bots. The difficulty is determined by the accuracy parameter. Any accuracy greater than 0.95 is considered hard (almost impossible mode), the bot will always choose the correct card according to the clue and if there are no more words related to the clue, it will still choose a card of their team.			
Attributes	Visibility	Data type	Name	Description
	Private	CardType	team	The operative's team.
	Private	Clue	currentClue	The current clue.
	Private	couble	accuracy	The bot's accuracy.
Methods	Visibility	Method Name		Description
	Public	BotOperativeStrategy (CardType team, double accuracy)		Constructor. Sets the operative's team and accuracy.
	Public	pickCard (List⟨Card⟩ cards, Bipartite bipartite)		Loops through all the cards and add the cards of their team into an ArrayList.

	Public	getClue()	Returns a string representing the current clue of a word.
	Public	setClue()	Sets a given string as the clue.

3.1.3.6 Class RandomOperativeStrategy

Class Name	randomOperativeStrategy		
Inherits From	implements OperativeStrategy		
Description	Random strategy; the operative picks cards at random.		
Methods	Visibility	Method Name	Description
	Public	pickCard (List⟨Card⟩ cards, Bipartite bipartite)	Picks a card at random.
	Public	getClue()	Returns the current clue.
	Public	setClue(Clue clue)	Sets the current clue.

3.1.3.7 Class Spymaster

Class Name	Spymaster			
Inherits From	extends Player			
Description	The implemented class for spymaster. Defines the basic functions and constructor that the spymaster strategies use.			
Attributes	Visibility	Data type	Name	Description
	Private	Board	board	The player's board they will play on.
	Private	SpyStrategy	spyStrat	The strategy the player will use.
	Private	Clue	currentClue	Current clue for the spymaster.
Methods	Visibility	Method Name		Description
	Public	Spymaster(CardType team)		Constructor. Instantiates the spymaster's team.
	Public	Spymaster(CardType team, Board board, OperativeStrategy strategy)		Constructor. Instantiates the spymaster's team and sets their board and strategy.
	Public	makeMove (Clue clue, Bipartite bipartite)		Gives a clue based on the operative strategy in use.

3.1.3.8 Class SpyStrategy

Class Name	SpyStrategy		
Inherits From	None		
Description	The interface for spymaster strategies. (Strategy pattern)		
Methods	Visibility	Method Name	Description
	Public	giveClue (List⟨Card⟩ cards, Bipartite bipartite)	Spymasters choose a clue based on the cards on the board.

3.1.3.9 Class RandomSpyStrategy

Class Name	randomSpyStrategy			
Inherits From	implements SpyStrategy			
Description	Random strategy; the spymaster gives clues at random.			
Attributes	Visibility	Data type	Name	Description
	Private	String[]	possibleClueWords	The list of clues the spymaster can give.
Methods	Visibility	Method Name		Description
	Public	giveClue (List⟨Card⟩ cards, Bipartite bipartite)		Gives a clue at random from the available clues.

3.1.3.10 Class SimpleSpyStrategy

Class Name	SimpleSpyStrategy			
Inherits From	implements SpyStrategy			
Description	Simple spy strategy: it randomly selects a word of their team and gives a random clue of the selected word. It won't try to combine words while giving the clues, so it will always return 1 word per clue.			
Attributes	Visibility	Data type	Name	Description
	Private	CardType	team	The spymaster's team.
Methods	Visibility	Method Name		Description
	Public	SimpleSpyStrategy(CardType team)		Constructor; sets the team.
	Public	giveClue (List⟨Card⟩ cards, Bipartite bipartite)		Selects a card from their team at random. Gives out a clue at random from the selected word

3.1.3.11 Class SmartSpyStrategy

Class Name	SmartSpyStrategy			
Inherits From	implements SpyStrategy			
Description	Allows the spy to choose a clue that includes many words on the board. The Spymaster will select a clue that is the most common amongst the words given to a particular player instead of just selecting a clue at random.			
Attributes	Visibility	Data type	Name	Description
	Private	CardType	team	The spymaster's team.
Methods	Visibility	Method Name		Description
	Public	SimpleSpyStrategy (CardType team)		Constructor; sets the team.
	Public	giveClue (List⟨Card⟩ cards, Bipartite bipartite)		Selects a card from their team at random. This method will search for the most optimal clue meaning the clue that is shared between the words belonging to the current player and return that specific clue

3.1.4 Package Util

The Util subsystem contains the Verbose class, which handles the models logging. Any logs generated by the model are sent to the Verbose class, which is a subject in the Observer design pattern. The Verbose class is a singleton, and when the Verbose objects log() method is called, it alerts any of its Observers to a change in its state, and allows them to retrieve the message being logged, in accordance with the Observer pattern.

3.1.4.1 Detailed Design Diagram

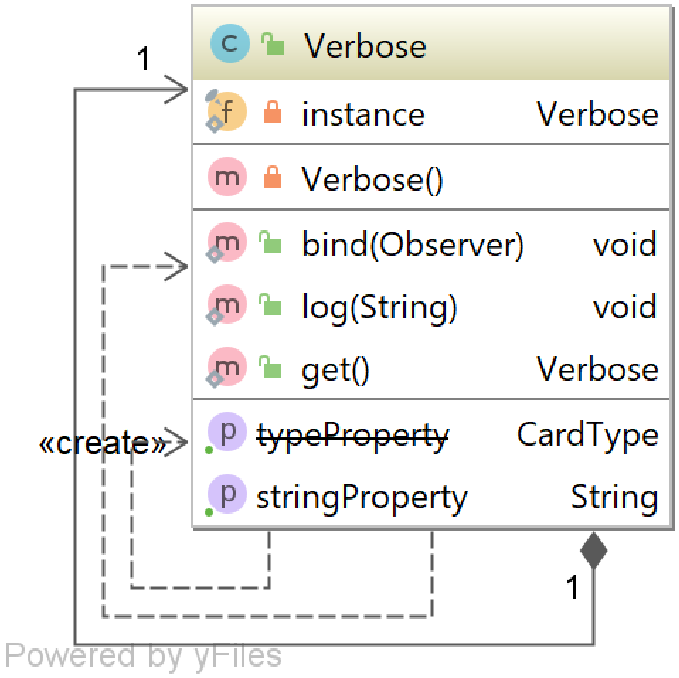

Figure 6: UML Diagram of Package Util in Module Model

3.1.4.2 Class Verbose

Class Name	Verbose			
Inherits From	extends Subject			
Description	Part of the Observer pattern. The model's interface to the view's VerboseView, used to log things which are optionally printed to a GUI.			
Attributes	Visibility	Data type	Name	Description
	Private	Verbose	instance	The verbose instance to which observers can be attached.
	Private	String	state	The message currently being displayed.
Methods	Visibility	Method Name		Description
	Private	Verbose()		Constructor.
	Public	bind(Observer o)		Attaches the observer to the instance so it will be alerted if the state changes.
	Public	log(String arg)		Sets the state to a new message and alerts observers.
	Public	get()		Returns the instance.

	Public	getStringProperty()	Returns the current message.
	Public	getTypeProperty()	Returns nothing, deprecated.

3.2 Subsystem Control

The Control subsystem encapsulates the parts of the system which enable users to control the game. It contains the event handler for button presses in on package and the classes used in the Command design pattern in another. When the user presses certain keys, the event handler creates a specific Command, and executes it, resulting in some function in the Model being called, likely changing the games state in the Model.

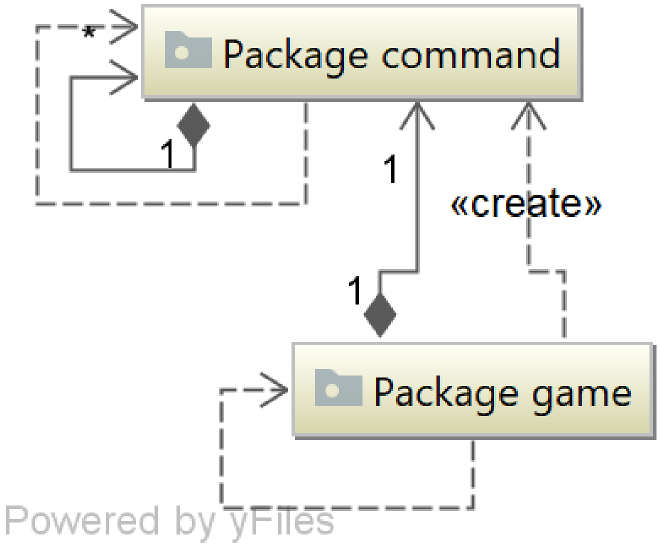

Figure 7: UML Diagram of Module Control

3.2.1 Package Command

The command package contains the classes which are a part of the Command design pattern. In our system the Controller and Model parts of the MVC architecture communicate using the Command pattern. The Controller makes function calls to the Model through classes which extend the Command interface. Because the Command pattern serves as an interface between two of the main subsystems, the Command related classes are grouped together in one package. The subsystem is comprised of the Command interface, and any classes extending it, as well as the CommandManager class which maintains a history of the Commands executed.

3.2.1.1 Detailed Design Diagram

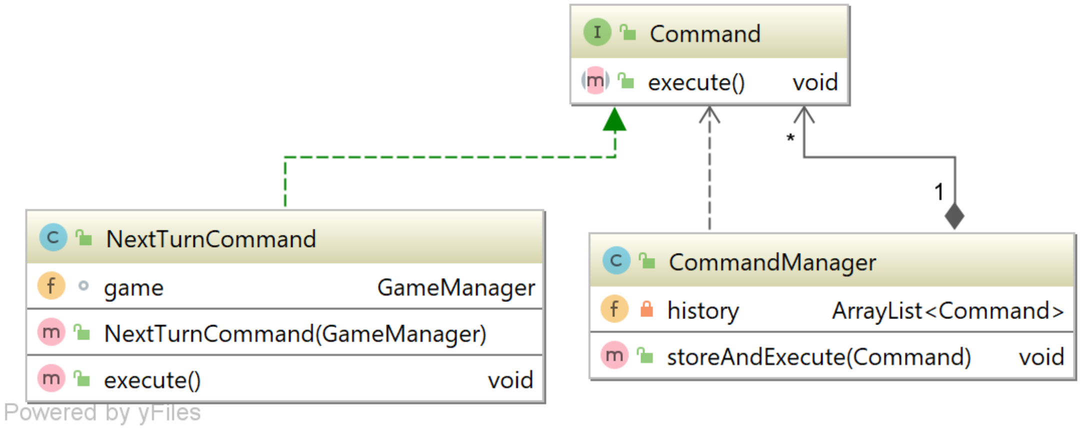

Figure 8: UML Diagram of Package Command in Module Control

3.2.1.2 Class Command

Class Name	Command		
Inherits From	None		
Description	Defines interface for commands (Command Pattern)		
Methods	Visibility	Method Name	Description
	Public	execute()	Executes command

3.2.1.3 Class CommandManager

Class Name	CommandManager			
Inherits From	None			
Description	Command manager to execute and store commands and also maintains a history of commands called.			
Attributes	Visibility	Data type	Name	Description
	Private	ArrayList⟨Command⟩	history	Object container responsible for containing history of commands
Methods	Visibility	Method Name		Description
	Public	storeAndExecute(Command cmd)		Stores the command in history then uses the execute() method to execute the command

3.2.1.4 Class NextTurnCommand

Class Name	NextTurnCommand			
Inherits From	Command			
Description	A Command to be used by control.GameHandler to tell the GameManager to run the next turn.			
Attributes	Visibility	Data type	Name	Description
	Private	GameManager	game	The part of the model that controls turn flow
Methods	Visibility	Method Name		Description
	Public	NextTurnCommand (GameManager game)		Constructor to set the game to the given input game
	Public	execute()		Override execute() command to do next turn.

3.2.2 Package Game

The Control's "Game" subsystem is mainly comprised of the event handler for user actions, such as key presses. The classes in this subsystem turn user interactions into Commands given to the Model, or manipulations of the View. This subsystem is the core of the Control part of the MVC architecture, serving as the method by which a user interacts with the Codenames game.

3.2.2.1 Detailed Design Diagram

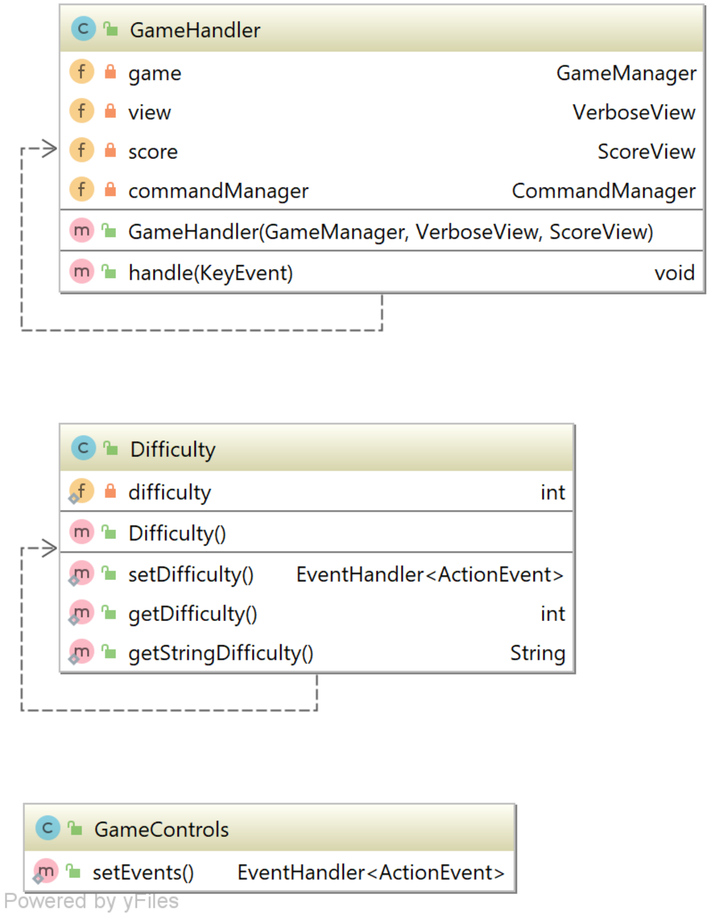
GameHandler
game GameManager
view VerboseView
score ScoreView
commandManager CommandManager
GameHandler(GameManager, VerboseView, ScoreView)
handle(KeyEvent) void
Difficulty
difficulty int
Difficulty()
setDifficulty() EventHandler<ActionEvent>
getDifficulty() int
getStringDifficulty() String
GameControls
setEvents() EventHandler<ActionEvent>
Powered by yFiles

Figure 9: UML Diagram of Package Game in Module Control

3.2.2.2 Class GameHandler

Class Name	GameHandler			
Inherits From	EventHandler⟨KeyEvent⟩			
Description	Main entry point of the program. Engine for running the Codenames game.			
Attributes	Visibility	Data type	Name	Description
	Private	GameManager	game	GameManager object responsible for initializing the game in the constructor.

	Private	VerboseView	view	VerboseView object responsible for initializing the view in the constructor.
	Private	ScoreView	score	ScoreView object responsible for initializing the score in the constructor.
	Private	CommandManager	commandManager	CommandManager object responsible for initializing the commandManager in the constructor as a new object.
Methods	Visibility	Method Name		Description
	Public	GameHandler(GameManager game, VerboseView view, ScoreView score)		Initializes the game, view, and score. Binds the commandManager to a new commandManager object. Shows the score scene.
	Public	handle(KeyEvent keyEvent)		When the user presses ENTER, the KeyHandler triggers the playerControl to play the next turn.

3.2.2.3 Class Difficulty

Class Name	Difficulty			
Inherits From	None			
Description	This class is responsible for setting the difficulty of the game.			
Attributes	Visibility	Data type	Name	Description
	Private	int	difficulty	Current difficulty. Default = 0.
Methods	Visibility	Method Name		Description

	Public	Difficulty()	Default constructor.
	Public	setDifficulty()	Event Handler that is responsible for setting the game difficulty based on input provided.
	Public	getDifficulty()	Retrieves the difficulty level of the game.
	Public	getStringDifficulty()	Returns a string representation of the level of difficulty of the game.

3.2.2.4 Class GameControls

Class Name	GameControls		
Inherits From	None		
Description	This class is responsible for handling the events regarding the Restart option or Quit. If Restart has been selected then the program will automatically restart the game and set up a new board.		
Methods	Visibility	Method Name	Description
	Public	setEvents()	Handles events. (Quit, Restart, About)
	Public	setAbout()	Handles the about command.

3.3 Subsystem View

The View subsystem encapsulates the parts of the system which control the GUI which the user sees. The ScoreView, VerboseView, and GameScene all display different parts of the state of the game Model, by binding themselves to Subject classes in the Model, in accordance with the Observer pattern. The ScoreView displays game state information. The VerboseView optionally shows logs of what is going on inside the Model. The GameScene is made up of 25 CardPanes which each observe Card objects in the model, changing the display when the Cards change state. By following the Observer pattern, the Model is allowed to exist without any View. The View provides a GUI, without effecting the Model subsystems.

3.3.1 Detailed Design Diagram

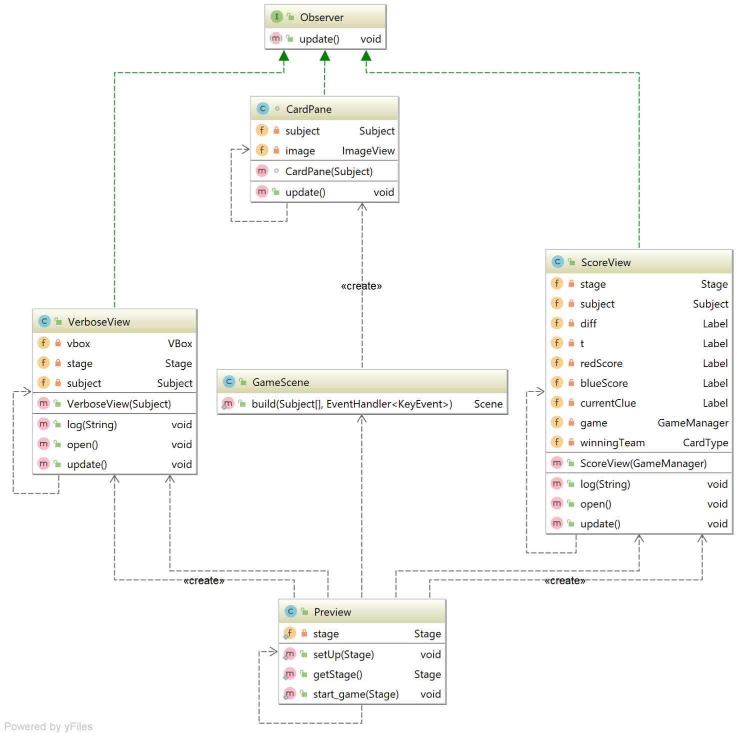

Figure 10: UML Diagram of Module View

3.3.2 Class Observer

Class Name	Observer		
Inherits From	None		
Description	Part of the Observer design pattern. Interface used for communication to the Subject.		
Methods	Visibility	Method Name	Description
	Public	update()	Update the state of the observer

3.3.3 Class VerboseView

Class Name	VerboseView			
Inherits From	Observer			
Description	Maintains graphical representation of the game play during each turn phase			
Attributes	Visibility	Data type	Name	Description

	Private	VBox	vbox	Object container responsible for containing the information to display
	Private	Stage	stage	Container supporting the vbox object during display
	Private	Subject	subject	The subject to bind to
Methods	Visibility	Method Name		Description
	Public	VerboseView(Subject s)		Constructor that binds to the subject
	Public	log(String arg)		Updates the text information to display with specified arg statement
	Public	open()		Displays the score window
	Public	update()		Calls log method and passes subject message to it

3.3.4 Class CardPane

Class Name	CardPane			
Inherits From	StackPane, Observer			
Description	Maintains graphical representation of a card during each game phase			
Attributes	Visibility	Data type	Name	Description
	Private	ImageView	image	Object responsible for loading character image files
	Private	Subject	subject	The subject to bind to
Methods	Visibility	Method Name		Description
	Protected	CardPane(Subject subject)		Constructor that binds to the subject
	Public	update()		Updates the ImageView object to display the associated character image file

3.3.5 Class GameScene

Class Name	GameScene		
Inherits From	None		
Description	Container for all CardPane objects. Factory for building graphical tree of all cards.		
Methods	Visibility	Method Name	Description

	Public	build(Subject[] subjects, EventHandler⟨KeyEvent⟩ handler)	Constructs the game scene containing all card nodes, and returns it to the invoker

3.3.6 Class ScoreView

Class Name	ScoreView			
Inherits From	Observer			
Description	Maintains a graphical representation of the score during the game play			
Attributes	Visibility	Data type	Name	Description
	Private	Stage	stage	Object container responsible for containing the attributes to display
	Private	Subject	subject	The subject to bind to
	Private	Label	diff	Object representing the difficulty
	Private	Label	t	Object representing the team's turn
	Private	Label	redScore	Object representing red team's score
	Private	Label	blueScore	Object representing blue team's score
	Private	Label	currentClue	Object representing the current clue of the turn
	Private	GameManager	game	Reference of the current game object to receive notifications from
	Private	CardType	winningTeam	Team currently in the lead
Methods	Visibility	Method Name		Description
	Public	ScoreView(GameManager game)		Constructor
	Public	log(String arg)		Updates the labels to reflect the current state of the game
	Public	open()		Displays the score window
	Public	update()		Calls log method and passes subject message to it

4 DYNAMIC DESIGN SCENARIOS

4.1 User Press Enter Scenario

The User Press Enter Scenario occurs when a user presses the Enter key, causing the Codenames game to play the next turn. The scenario shows the use of the Command Design patter between the Controller and Model parts of the system. When the user presses enter, GameHandler class of the Controller is called to handle the event. The GameHandler class creates a new instance of a Command to trigger the next turn, and then sends it to the CommandManager to record it, and execute the command. When the command is executed, it calls upon the Model's GameManager class to do the next turn.

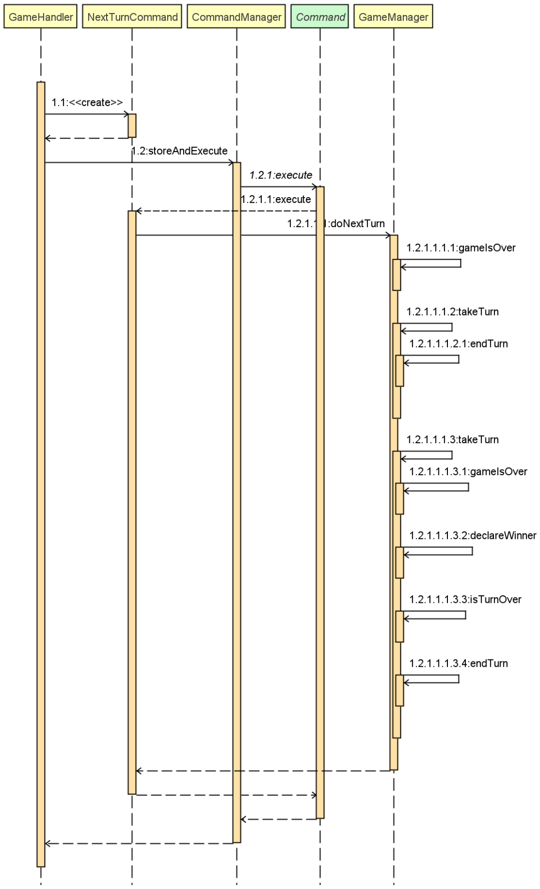

Figure 11: Sequence Diagram of User

4.2 Spymaster's Turn Scenario

A major part of the Codenames game is the Spymasters giving clues. In our design the simulated players are objects, and the GameManager calls their makeMove() functions to trigger them to play the game. In the case of the Spymaster, the GameManager tells the Spymaster to make a move, expecting it to return a clue. Following the Strategy design pattern, a Spymaster may implement different strategies for generating this clue. The Spymaster calls on its specific SpyStrategy object to create the clue returning the clue to the GameManager.

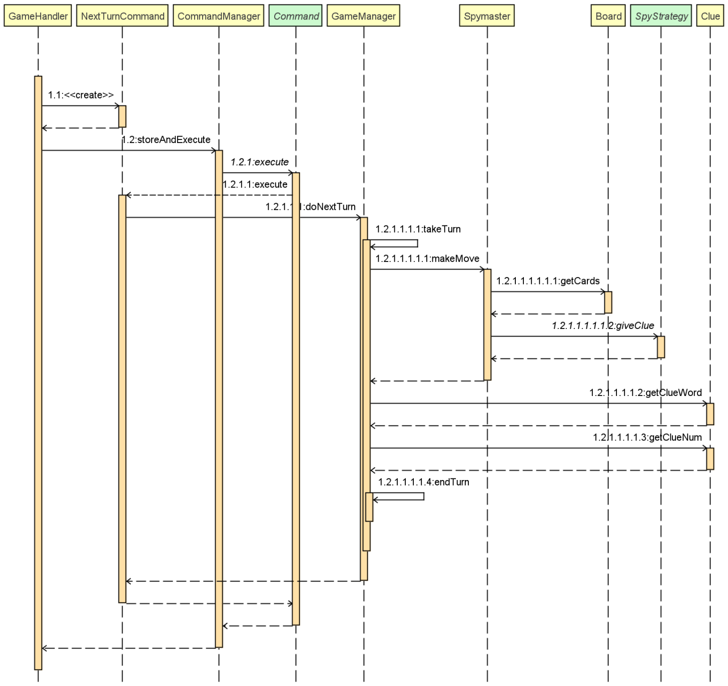

Figure 12: Sequence Diagram of Spymaster

4.3 Operative's Turn Scenario

Once a clue is given by a Spymaster, it is the Operative players turn to guess the cards on the board associated with that clue. In our system the GameManager calls an Operative object's makeMove, passing the given clue as a parameter. The Operative's play style is implemented using the Strategy design pattern. The Operative's makeMove method first telling its concrete strategy what the current clue is, and then by calling the OperativeStrategy's pickCard() class, passing the possible card choices as a parameter. The concrete strategy then returns one of the choices, and the Operative returns the choice to the GameManager.

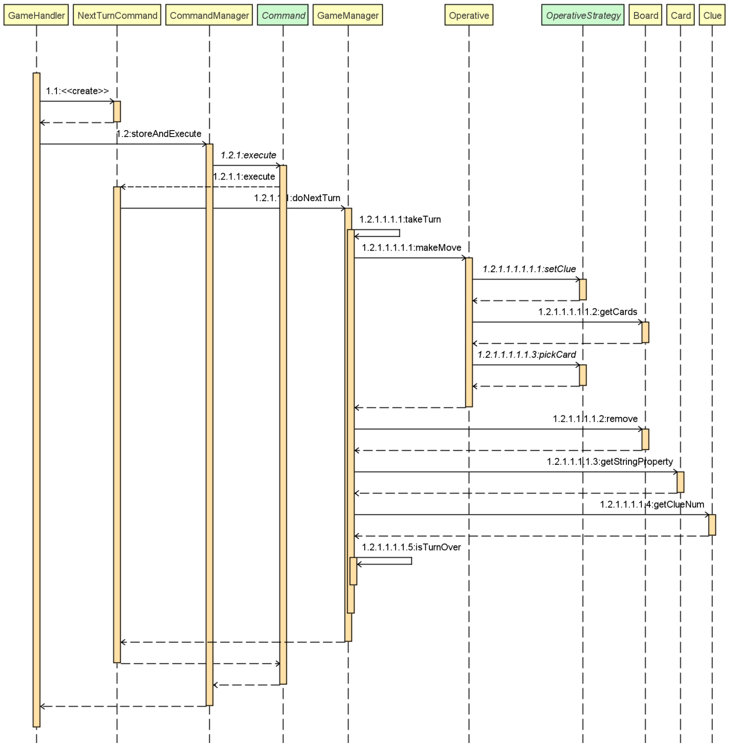

Figure 13: Sequence Diagram of Operative