Requirements Document

Team PJA

January - April 2019

Table 1: Team Members

Name	ID Number
Souheil Al-Awar	26558038
David Boivin	40004941
Benson Chan	40046280
Annes Cherid	40038453
Carl Neil Cortes	40016567
Gaoshuo Cui	40085020
Robert Laviolette	27646666
Karim Loulou	40027203
Ke Ma	26701531
Kevin McAllister	40031326
Yogesh Nimbhorkar	40093384

Contents

1 System

1.1 Purpose

The purpose of this document is to showcase and explain the separate parts of the group project for COMP 354. This project's main goal is illustrate the development of the game Codenames in order to experience the life cycle of team-driven software creation under strict time constraints. It will be based on the design of architecture, functionalities, and implementation. The architecture section will describe the ideas behind the software structure that is selected for the game development, and a class diagram (UML) will be provided for this. The functionality section will include details of how the users interact with the system by introducing a series of use cases. Finally, the implementation section will describe the use of each class and object that are created inside the program, and how they work with each other to achieve the goal. This document is presented by all members of team PJA as a blueprint used for the entire game development process.

1.2 Context

Codenames was created in 2015, and is currently still rated as the number 1 party game on BoardGameGeek (BGG)'s website. It has at least 25 awards to its name, including the 2016 Spiel des Jahres (Game of the year) and several Golden Geek Best Party and Family game awards.

One of the course objectives of COMP 354 is to learn the process of developing software in a team project. Our team is to adapt this game from the form of a traditional card game to a digital version, in order to allow more portability of the game. This should increase accessibility and create a wider audience of play.

For the development of Codenames we are using Java as the programming language. It has a basic graphical user interface and is built using the Model-View-Controller architecture. Eclipse is the integrated development environment used for our development, with JUnit used for unit testing.

1.3 Business Goals

The intent of the creation of the Codenames digital adaptation can be summarized in a few concise business goals:

- To fulfill the requirements set by the COMP 354 Codenames project outline
- To create the Codenames game over three iterations, within the deadlines given to us
- To experience the life cycle of developing software in a team-based environment in order to improve our abilities in working as a team under strict constraints and to gain valuable experience usable in a real-world workplace environment.

1.4 Game Concept

Codenames is a game of guessing which codenames (words) in a set are related to a hint-word given by another player. Players split into two teams: red and blue. One player of each team is selected as the teams spymaster; the others are field operatives. Twenty-five code name cards, each bearing a word, are laid out in a 55 rectangle grid, in random order. A number of these words represent red agents, a number represent blue agents, one represents an assassin, and the others represent innocent bystanders.

The teams spymasters are given a randomly-dealt map card showing a 5-by-5 grid of 25 squares of various colors, each corresponding to one of the code name cards on the table. Teams take turns. On each turn, the appropriate spymaster gives a verbal hint about the words on the respective cards. Each hint may only consist of one single word and a number. The hints word can be chosen freely, as long as it is not (and does not contain) any of the words on the code name cards still showing at that time. Code name cards are covered as guesses are made. The hints number tells the field operatives how many words in the grid are related to the word of the clue. It also determines the maximum number of guesses the field operatives may make on that turn, which is the hints number plus one. Field operatives must make at least one guess per turn, risking a wrong guess and its consequences. They may also end their turn voluntarily at any point thereafter.

After a spymaster gives the hint with its word and number, their field operatives make guesses about which code name cards bear words related to the hint and point them out, one at a time. When a code name card is pointed out, the spymaster covers that card with an appropriate identity card, a blue agent card, a red agent card, an innocent bystander card, or the assassin card as indicated on the spymasters map of the grid. If the assassin is pointed out, the game ends immediately, with the team who identified him losing. If an agent of the other team is pointed out, the turn ends immediately, and that other team is also one agent closer to winning. If an innocent bystander is pointed out, the turn simply ends.

The game ends when all of one teams agents are identified (winning the game for that team), or when one team has identified the assassin (losing the game).

2 Domain Concepts

The following figures show the real-life structure of the Codenames board game, illustrating domain concepts (enclosed in squares) and relationships (represented by arrows) through a domain model. Figure 1 intentionally focuses on semantics to explain the game in simple terms. Figure 2 adds upon Figure 1 by naming the attributes of the domain concepts.

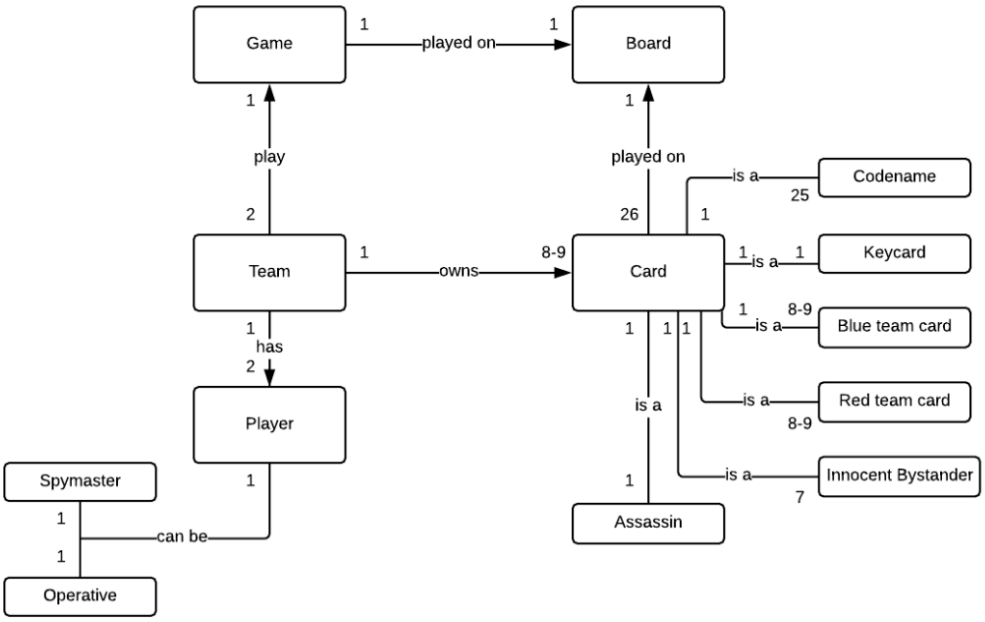


Figure 1: Domain Diagram

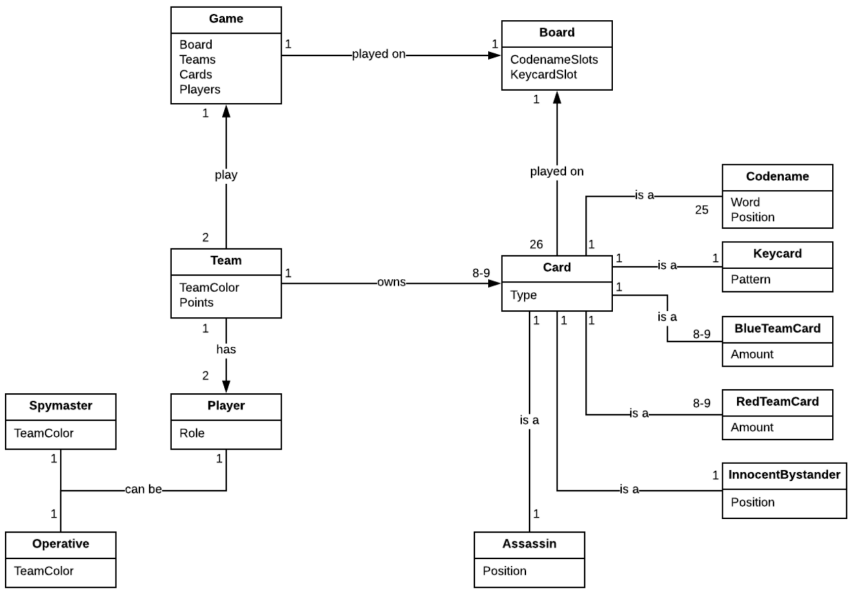


Figure 2: Domain model with attributes

3 Use Cases

3.1 Overview

The Codename game is divided into 3 Use Cases:

- User (3.1.1)
- Spymaster(3.1.2)
- Operator (3.1.3)

Each of these use cases defines the correlation between each actor in relation to the Codenames board game (spymaster, operative, user).

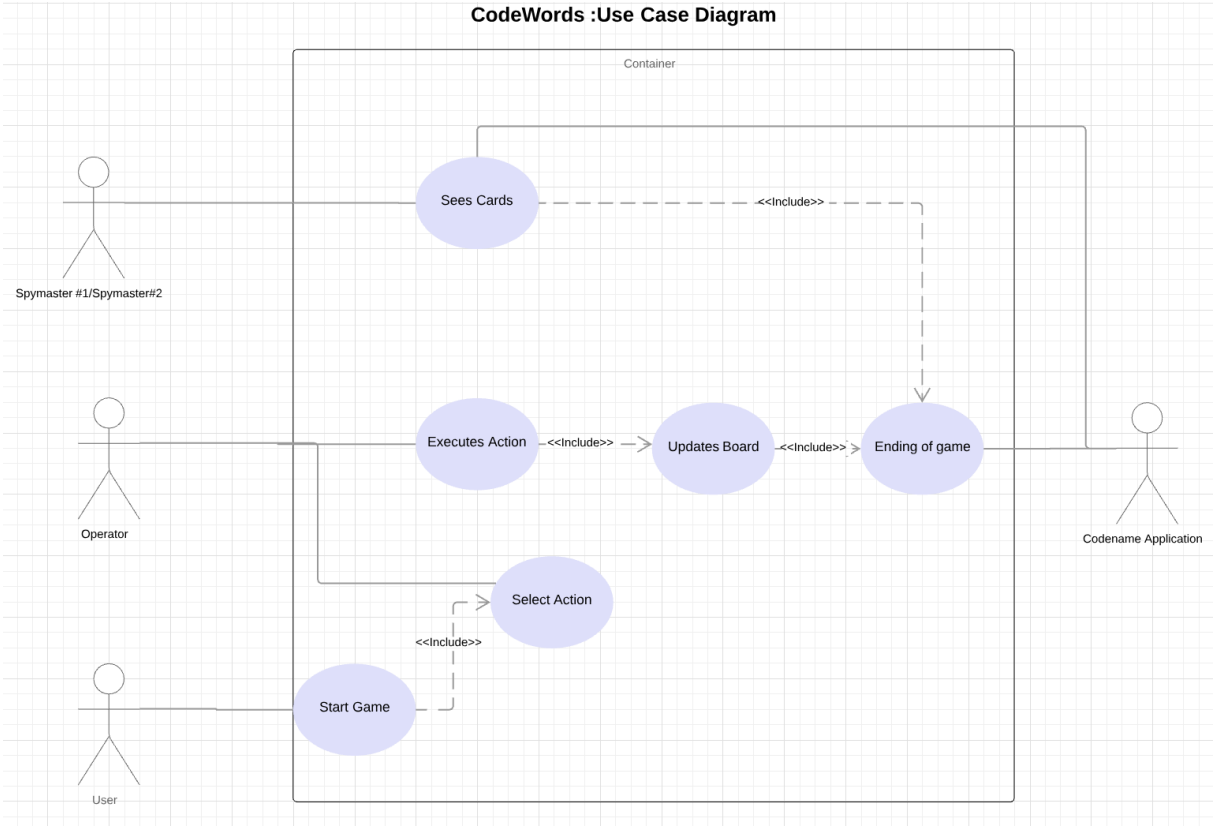


Figure 3: General Use Case Diagram

3.1.1 Use Case: User

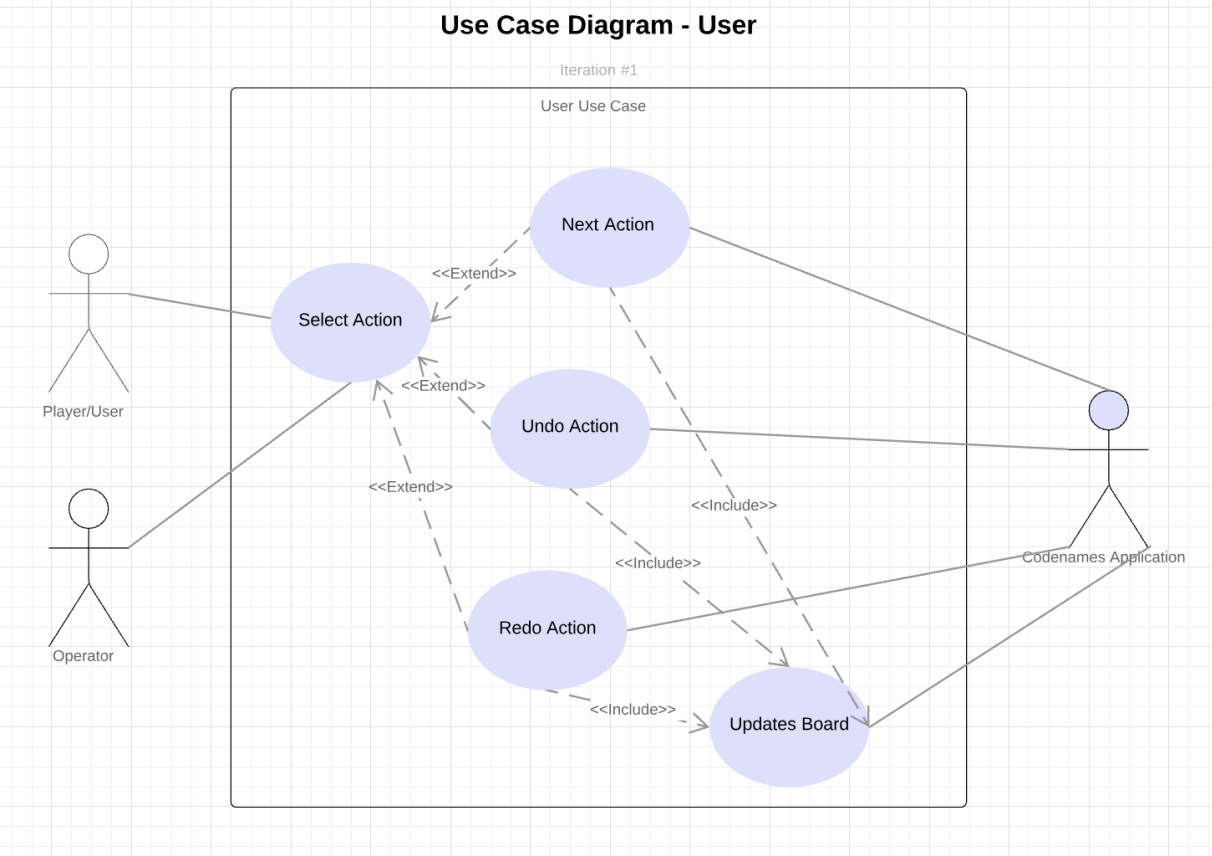


Figure 4: User Use Case Diagram

3.1.2 Use Case: Spymaster

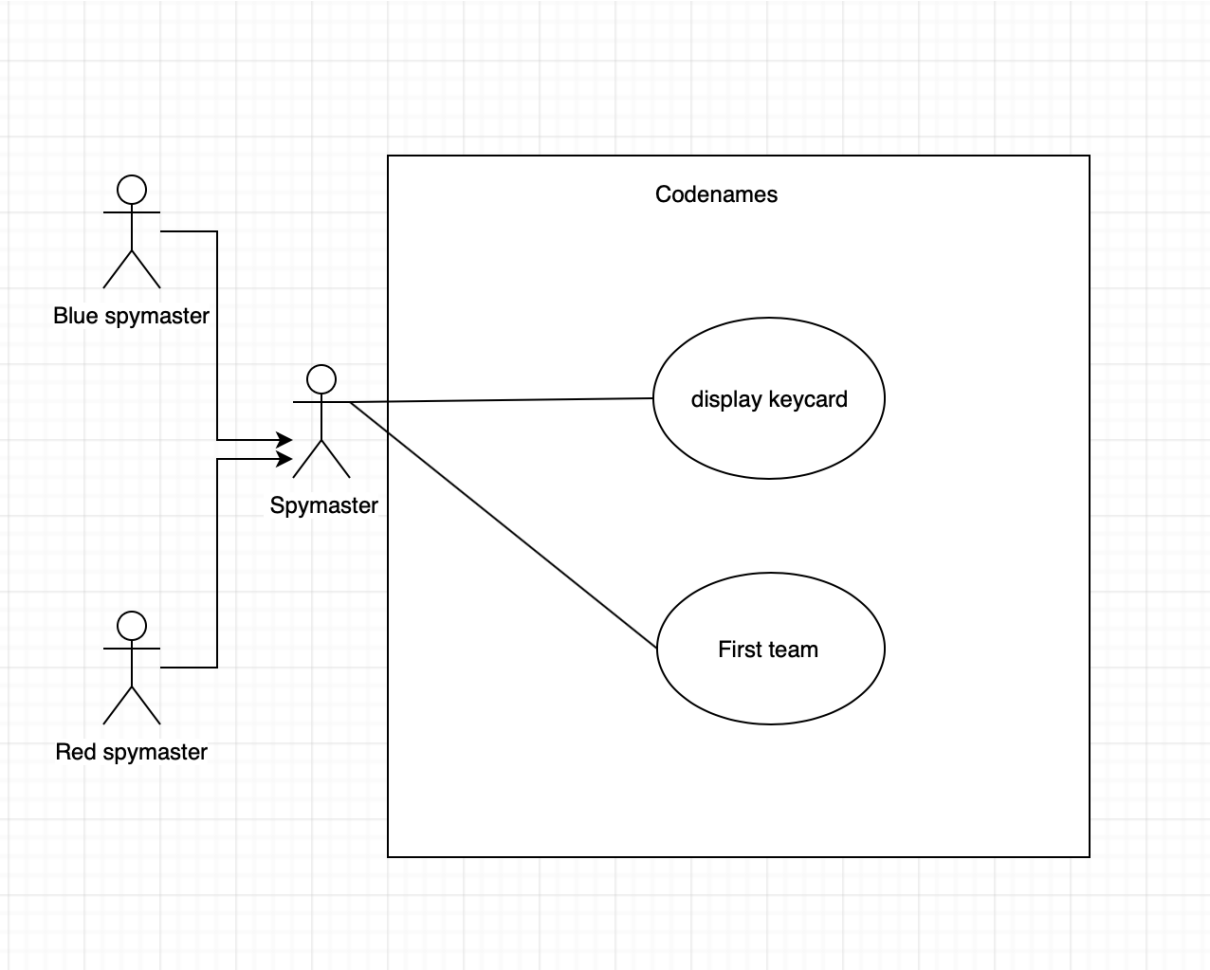


Figure 5: Spymaster Use Case Diagram

3.1.3 Use Case: Operator

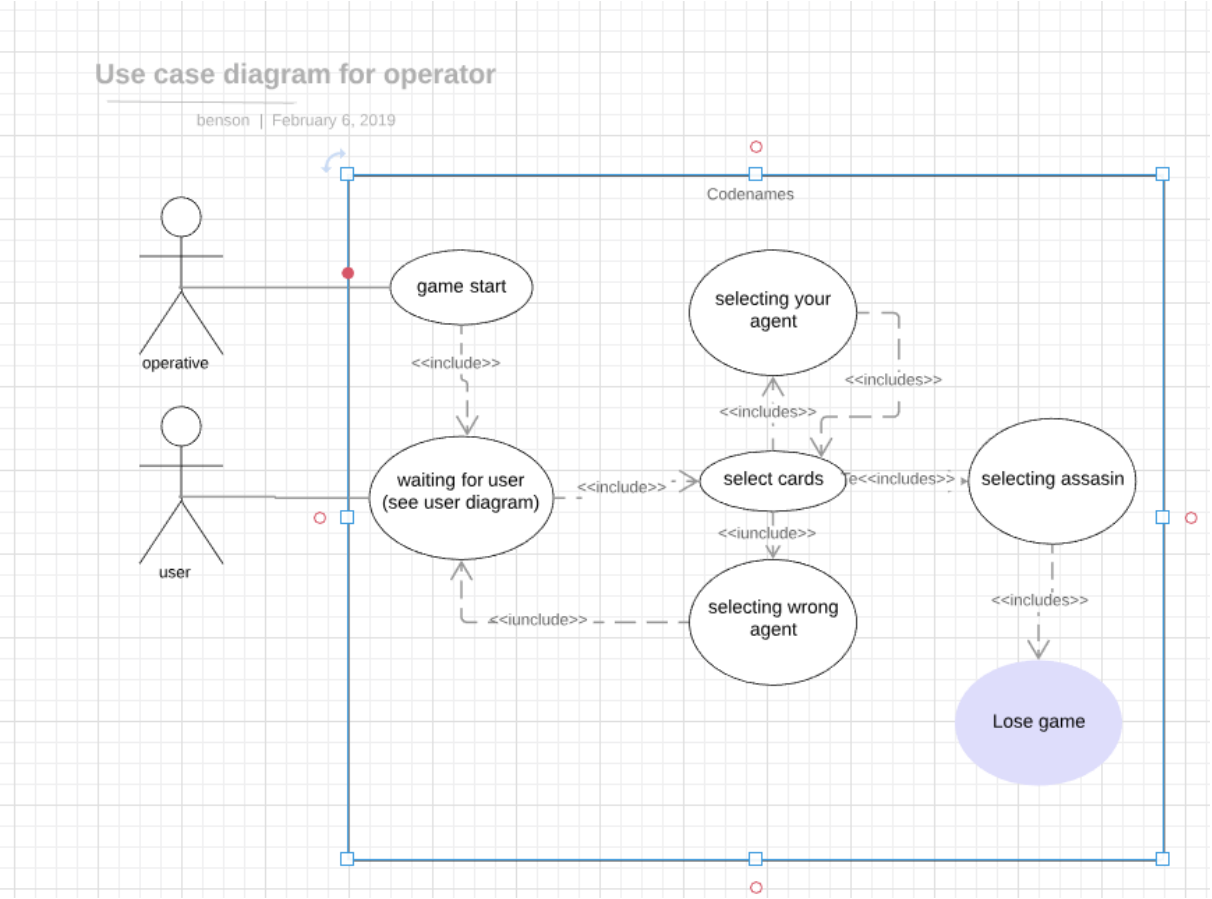


Figure 6: Operator Use Case Diagram

3.2.1 Use Case Diagram Table: User

Description	The user enters the game and starts playing
Actors	User and Operator
Pre-Conditions	• Player must already know the rules of the game
Flow of events	
Basic Path	1. The user selects the option "Start Game". 2. Then the player waits for operator to "Start Game". 3. The board is initialized. 4. The user can select Next Action and the Operator will select random cards for each team. 5. The board updates as the game progresses. 6. Steps 4 and 5 are repeated until a winner is declared.
Alternative Paths	• Instead of pressing the "Next Action" button, the user can choose the "Undo" button to undo the last action done by the operator. • Instead of pressing the "Next Action" button, the user can choose the "Redo" button to redo the last action undone .
Post conditions	• The board has been initialized. • The Codename cards are placed randomly onto the 5x5 board face down. • Player from team blue and team red can play.
Related Use Cases	
Used use Cases	None
Extending Use Cases	The Operator Use Case

3.2.2 Use Case Diagram Table: Spymaster

Description	The spymaster needs to view the keycard in this game
Actors	Spymaster
Pre-Conditions	• The board has been initialized. • The Codename cards are placed randomly onto the 5*5 board face down.
Flow of Events	
Basic Path	1. For each team of spymaster, click “view keycard” button to look at the keycard which according to the randomly generated board. 2. View which team starts first. 3. Close the keycard window by pressing the close button. 4. Wait until game ends. 5. No hints in iteration 1.
Alternative Paths	None
Post-Conditions	None
Related Use Cases	None
Used Use Cases	None
Extending Use Cases	None

3.2.3 Use Case Diagram Table: Operator

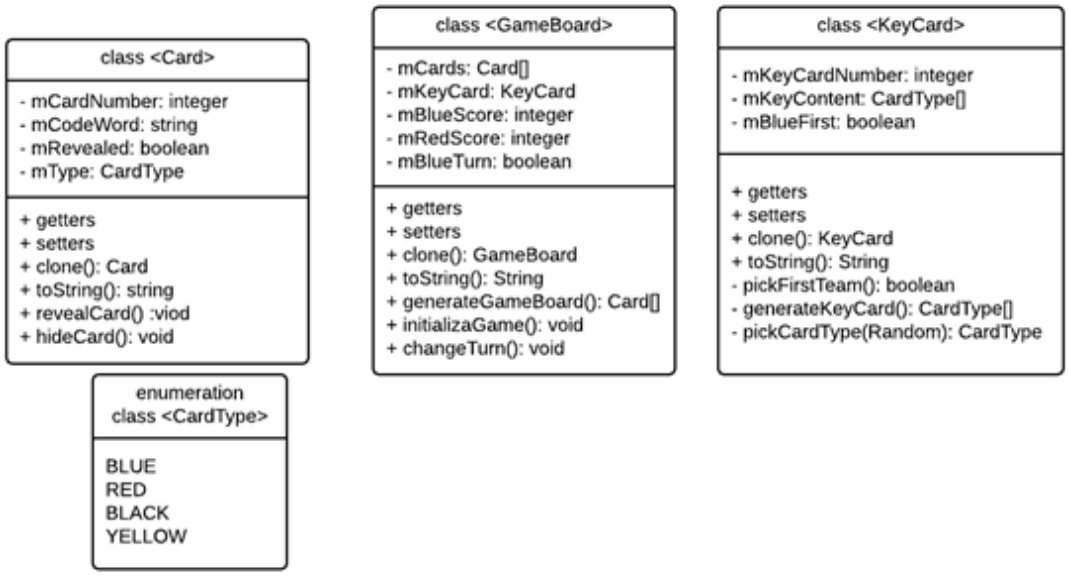
Description	As an operative, the goal is contact the other agents of your team based on the clues your spymaster gives you.
Actors	Operative (player)
Pre-Conditions	None
Flow of Events	
Basic Path	1. Operatives clicks start game 2. Waits for USER (spymaster) to act first 3. Receive a clue from spymaster and select a card (Random / Next Available) based on the clues the spymaster has given until you pick a bystander agent, an assassin, or the opposite agent. 4. Let opposite operative team play their turn 5. Repeat step 3 and 4 until first one to contact all their agents, or until someone contacts the assassin*. *Contacting the assassin results in losing immediately.
Alternative Paths	None
Post-Conditions	• Pick agents from your team to continue your turn.
Related Use Cases	Spymasters
Used Use Cases	Spymaster's clues
Extending Use Cases	None

4 MVC architecture

4.1 Model

The game has objects that reflect real-world things. The model will include some classes that describe the objects that we use in the game, such as the game board, card and key card. Game board is the 5x5 rectangular grid, it is the base of the game.According to codenames game rule, cards have different properties, blue card is for blue team, red car is for red team, black card is the assassin, yellow card is for bystander. Each card has different status, such as revealed or hidden. The key card is for the card that is visible for spymaster only, which is used for spymasters to give clues to operatives, it is randomly generated. All the words that may appear in the game are generated from out word database.

4.1.1 class Diagram



4.1.2 class <Card>

Class name	Card			
Description	Card object that stores data for each Codename			
Attributes	Visibility	Data type	Name	Description
	private	integer	mCardNumber	stores the number of this card.
	private	String	mCodeWord	Stores the code word on this card.
	private	boolean	mRevealed	Stores the status of this card (revealed or hidden).
	private	CardType	mType	Stores the type of this card.
Method	Visibility	Method Name	Description	
	public	Card(int,String,CardType)	Constructor	
	public	Card(int,String,CardType,String[])	Constructor	
	public	int getCardNumber()	getter	
	public	String getCodeWord()	getter	
	public	boolean isRevealed()	To check if the card is revealed	
	public	CardType getType()	getter	
	public	revealCard()	To reveal a card.	
	public	hideCard()	To hide a card.	

4.1.3 class <CardType>

Enumeration on 4 types of card:

BLUE : blue team
RED : red team
BLACK : assassin
YELLOW : bystander

4.1.4 class <GameBoard>

Class Name	GameBoard			
Description	GameBoard object to store cards, copy of keycard and the GameObserver.java object			
Attributes	Visibility	Data type	Name	Description
	private	ArrayList<Card>	mCards	stores the cards.
	private	KeyCard	mKeyCard	stores the key card.
	private	GameObserver	mGameObserver	stores the observation of current status.
Methods	Visibility	Method Name	Description	
	public	GameBoard(KeyCard, Outbox)	Constructor.	
	public	Card getCard(int)	getter.	
	public	ArrayList<Card> getBoard()	getter.	
	public	KeyCard getKeyCard()	getter.	
	public	ArrayList<Card> getCards()	getter.	
	private	ArrayList<Card> generateGameBoard()	To generate the random 25 cards game board.	

4.1.5 class <Keycard>

Class Name	KeyCard			
Description	KeyCard object that stores the board layout and the starting team			
Attributes	Visibility	Data type	Name	Description
	private	integer	mKeyCardNumber	Stores the number of this key card.
	private	CardType[]	mKeyContent	Stores content of this key card.
	private	boolean	mBlueFirst	Stores if the blue team goes first.
Methods	Visibility	Method Name	Description	
	public	KeyCard(int)	Constructor.	
	public	boolean getBlueFirst()	getter.	
	public	CardType getCardType(int)	getter.	
	public	CardType[] getKeyContent()	getter.	
	public	setKeyCardNumber(int)	setter.	
	public	setBlueFirst(boolean)	setter.	
	public	setKeyContent(CardType[])	setter.	
	public	String toString()	override toString to display information.	
	private	boolean pickFirstTeam()	To check which team goes first.	
	private	CardType[] generateKeyCard()	To generate key cards.	
	private	CardType pickCardType(Random)	To generate a random card type storing into key card.	

4.1.6 Word Database

The wordDatabase file is used for the following word that may be generated in the game:

Acne, Acre, Addendum, Advertise, Aircraft, Aisle, Alligator, Alphabetize, America, Ankle, Apathy, Applause, Applesauce, Application, Archaeologist, Aristocrat, Arm, Armada, Asleep, Astronaut,Athlete, Atlantis, Aunt, Avocado, Baby-Sitter, Backbone, Bag, Baguette, Bald, Balloon, Banana, Banister, Baseball, Baseboards, Basketball , Bat, Battery , Beach, Beanstalk, Bedbug, Beer, Beethoven, Belt, Bib, Bicycle, Big, Bike, Billboard, Bird, Birthday, Bite, Blacksmith, Blanket, Bleach, Blimp, Blossom, Blueprint, Blunt, Blur, Boa, Boat, Bob, Bobsled, Body , Bomb, Bonnet, Book, Booth, Bow-tie, Box, Boy , Brainstorm, Brand, Brave, Bride, Bridge, Broccoli, Broken ,Broom, Bruise, Brunette, Bubble, Buddy, Buffalo, Bulb, Bunny, Bus, Buy, Cabin, Cafeteria, Cake, Calculator, Campsite, Can, Canada, Candle, Candy, Cape, Capitalism, Card, Cardboard,Cartography, Car , Cd, Ceiling, Cell, Century, Chair , Chalk, Champion, Charger, Cheerleader, Chef, Chess, Chew, Chicken, Chime, China, Chocolate, Church, Circus, Clay, Cliff, Cloak, Clockwork, Clown, Clue ,Coach, Coal, Coaster, Cog, Cold, College, Comfort, Computer, Cone, Constrictor, Continuum, Conversation, Cook, Coop, Cord, Corduroy, Cot, Cough, Cow, Cowboy, Crayon, Cream, Crisp, Criticize, Crow, Cruise, Crumb, Crust, Cuff, Curtain, Cuticle, Czar, Dad, Dart, Dawn, Day, Deep, Defect, Dent, Dentist, Desk, Dictionary, Dimple, Dirty, Dismantle, Ditch, Diver, Doctor, Dog, Doghouse, Doll, Dominoes, Door, Dot, Drain, Draw, Dream, Dress, Drink, Drip, Drums, Dryer, Duck, Dump, Dunk, Dust, Ear, Eat, Ebony, Elbow, electricity, Elephant, Elevator, Elf, Elm, Engine, England, Ergonomic, Escalator, Eureka, Europe, Evolution, Extension,

Eyebrow, Fan, Fancy, Fast, Feast, Fence, Feudalism, Fiddle, Figment, Finger, Fire, First, Fishing, Fix, Fizz, Flagpole, Flannel, Flashlight, Flock, Flotsam, Flower, Flu, Flush, Flutter, Fog, Foil, Football, Forehead, Forever, Fortnite, Furnace, Freckle, Freight, Fringe, Frog, Frown, Gallop, Game, Garbage, Garden, Gasoline, Gem, Ginger, Gingerbread, Girl, Glasses, Goblin, Gold, Goodbye, Grandpa, Grape, Grass, Gratitude, Gray, Green, Guitar, Gum, Gumball, Hair, Half, Handle, Handwriting, Hang, Happy, Hat, Hatch, Headache, Heart, Hedge, Helicopter, Hem, Hide, Hill, Hockey, Homework, Honk, Hopscotch, Horse, Hose, Hot ,House, Houseboat, Huge, Humidifier, Hungry, Hurdle, Hurt, Hut, Ice, Implode, Inn, Inquisition, Intern, Internet, Invitation, Ironic, Ivory, Ivy, Jade, Jeans, Japan, Jelly, Jet, Jig, Jog, Journal, Jump, Key, Killer, Kilogram, King, Kitchen, Kite, Knee, Kneel, Knife, Knight, Koala, Lace, Ladder, Ladybug, Lag, Landfill, Lap, Laugh, Laundry, Law, Lawn, Lawnmower, Leak, Leg, Letter, Level, Lifestyle, Ligament, Light, Lightsaber, Lime, Lion, Lizard, Log, Loiterer, Lollipop, Loveseat, Loyalty, Lunch, Lunchbox, Lyrics, Machine, Macho, Mailbox, Mammoth, Mark, Mars, Mascot, Mast, Matchstick, Mate, Mattress, Mess, Mexico, Midsummer, Mine, Mistake, Modern, Mold, Mom, Monday, Money, Monitor, Monster, Mooch, Moon, Moth, Mop, Motorcycle, Mountain, Mouse, Mower, Mud, Music, Mute, Nature, Negotiate, Neighbour,Nest, Neutron, Niece, Night, Nightmare, Nose, Oar, Observatory, Office, Oil, Old, Olympian, Opaque, Opener, Orbit, Organ, Organize, Outer, Outside, Ovation, Overture, Pail, Paint,Pajamas, Palace, Pants, Paper, Park, Parody, Party, Password, Pastry, Pawn, Peer, Pen, Pencil, Pendulum, Penis, Penny, Pepper, Personal, Philosopher, Phone, Photograph, Piano, Picnic, Pigpen, Pillow, Pilot, Pinch, Ping, Pinwheel, Pirate, Plaid, Plan, Plank, Plate, Platypus, Playground, Plow, Plumber, Pocket, Poem, Point, Pole, Pomp, Pong, Pool, Popsicle, Population, Portfolio, Positive, Post, Princess, Procrastinate, Protestant, Psychologist, Publisher, Punk, Puppet, Puppy, Push, Puzzle, Quarantine, Queen, Quicksand, Quiet, Race, Radio, Raft, Rag, Rainbow, Rainwater, Random, Ray, Recycle, Red, Regret, Recycle, Reimburse, Retaliate, Rib, Riddle, Rim, Rink, Roller, Room, Rose, Round, Roundabout, Rung, Runt, Rut, Sad, Safe, Salmon, Salt, Sandbox, Sandcastle, Sandwich , Sash, Satellite, Scar, Scared, School, Scoundrel, Scramble, Scuff, Seashell, Season, Sentence, Sequins, Set, Shaft, Shallow, Shampoo, Shark, Sheep, Sheets, Sheriff, Shipwreck, Shirt, Shoelace, Short, Shower, Shrink, Sick, Siesta, Silhouette, Singer, Sip, Skate, Skating, Ski, Slam, Sleep, Sling, Slow, Slump, Smith, Sneeze, Snow, Snuggle, Song, Space, Spare, Speakers, Spider, Spit, Sponge, Spool, Spoon, Spring, Sprinkler, Spy, Square, Squint, Stairs, Standing, Star, State, Stick, Stockholder, Stoplight, Stout, Stove, Stowaway, Straw , Stream, Streamline, Stripe, Student, Sun, Sunburn, Sushi, Swamp, Swarm, Sweater, Swimming, Swing, Tachometer, Talk, Taxi, Teacher, Teapot, Teenager, Telephone, Ten, Tennis, Thief, Think, Throne, Through, Thunder, Tide, Tiger, Time, Tinting, Tiptoe, Tiptop, Tired, Tissue, Toast, Toilet, Tool, Toothbrush, Tornado, Tournament, Tractor, Train, Trash, Treasure, Tree, Triangle, Trip, Truck, Tub, Tuba, Tutor, Television, Twang, Twig, Twitterpad, Type, Unemployed, Upgrade, Vest, Vision, Wag, Water, Watermelon, Wax, Wedding, Weed, Welder, Whatever, Wheelchair, Whiplash, Whisk, Whistle, White, Wig, Will, Windmill, Winter, Wish, Wolf, Wool, World, Worm, Wristwatch, Yardstick, Zamboni, Zen, Zero, Zipper,Zone, Zoo

4.2 View

While the Model handles the parts of the game, the View must make it so that the users actually know whats going on. The View includes several classes whose goals are output.

4.2.1 class <BoardPane>

Class Name	BoardPane			
Description	Root of Graphical User Interface			
Attributes	**Visibility**	**Data Type**	**Name**	**Description**
	private	ArrayList<CardView>	mCVList	
	private	Inbox	mInbox	Used to send messages to the Controller
	private	ControlBar	mControl	
	private	HQPane	mHQ	
	private	FieldPane	mField	
	private	KeycardPopup	mPopup	

Methods	**Visibility**	**Method Name**	**Description**
	public	BoardPane(List<Card>, KeyCard, Inbox)	
	public	update()	

Inner Classes	**Visibility**	**Class Name**	**Implements**	**Description**
	private	CardHandler	EventHandler<ActionEvent>	
	private	KeycardHandler	EventHandler<ActionEvent>	
	private	NewGameHandler	EventHandler<ActionEvent>	Sends a new game message through inbox
	private	UndoHandler	EventHandler<ActionEvent>	Sends an undo message through inbox
	private	RedoHandler	EventHandler<ActionEvent>	Sends a redo message through inbox
	private	NextHandler	EventHandler<ActionEvent>	Sends a next message through inbox

4.2.2 class <CardView>

Class Name	CardView			
Description	UI element which represents the cards of CodeNames			
Attributes	**Visibility**	**Data Type**	**Name**	**Description**
	private	int	mID	The card's ID

Methods	**Visibility**	**Method Name**	**Description**
	public	CardView(int, String)	Constructor
	public	getCardID()	Getter
	public	changeColor(CardType)	Change the color of the card

Inner Classes	**Visibility**	**Class Name**	**Implements**	**Description**
	private	EnterHandler	EventHandler<MouseEvent>	Handles when the mouse enters into the card
	private	ExitHandler	EventHandler<MouseEvent>	Handles when the mouse leaves the card

4.2.3 class <ControlBar>

Class Name	ControlBar			
Description	A container which has all the user control buttons for the program.			
Attributes	Visibility	Data Type	Name	Description
	private	ControlButton	mNew	
	private	ControlButton	mNext	
	private	ControlButton	mUndo	
	private	ControlButton	mRedo	
	private	ControlButton	mKeycard	
Methods	Visibility	Method Name		Description
	public	ControlBar()		Constructor
	public	setNewGameHandler(EventHandler<ActionEvent>)		Setter
	public	setNextHandler(EventHandler<ActionEvent>)		Setter
	public	setUndoHandler(EventHandler<ActionEvent>)		Setter
	public	setRedoHandler(EventHandler<ActionEvent>)		Setter
	public	setKeycardHandler(EventHandler<ActionEvent>)		Setter

Inner Class	Class Name	ControlButton			
	Description	For consistency across all buttons in control bar			
	Attributes	Visibility	Data Type	Name	Description
	Methods	Visibility	Method Name		Description
		public	ControlButton(String)		Constructor
		private	defaultStyle()		Style helper
		private	pressedStyle()		Style helper
		private	enteredStyle()		Style helper
	Inner Classes	Visibility	Class Name	Implements	Description
		private	ExitHandler	EventHandler<MouseEvent>	Handles when the mouse exits a buttons region
		private	PressedHandler	EventHandler<MouseEvent>	Handles when the button is pressed.
		private	ReleaseHandler	EventHandler<MouseEvent>	Handles when the button is released

4.2.4 class <FieldPane>

Class Name	FieldPane			
Description	Contains the card elements of the game in a grid layout			
Attributes	Visibility	Data Type	Name	Description
	private	TreeMap<Integer, CardView>	mCardElements	
Methods	Visibility	Method Name	Description	
	public	FieldPane(List<CardView>)	Constructor	
	public	changeCardColor(int, CardType)	Changes the card color	

4.2.5 class <HQPane>

Class Name	HQPane			
Description	Provides all the game information from the current keyword to the score of each team			
Attributes	**Visibility**	**Data Type**	**Name**	**Description**
	private	TeamPane	mBlueTeam	
	private	TeamPane	mRedTeam	
Methods	**Visibility**	**Method Name**		**Description**
	public	HQPane(boolean)		Constructor
	public	setClue(boolean, String)		Sets the clue to the given team
	public	setScore(boolean, int)		Changes the score of the given team
	public	setTurn(boolean)		Changes the color on the teams to display turn
Inner Classes	**Class Name**	TeamPane		
	Description	Container which represents each team in the game		
	Attributes	**Visibility**	**Data Type**	**Name**	**Description**
		private	boolean	mIsBlue	
		private	Clue	mClue	
		private	Score	mScore	
	Methods	**Visibility**	**Method Name**		**Description**
		public	TeamPane(boolean, boolean)		Constructor
		public	setClue(String)		Setter
		public	setScore(int)		Setter
		public	setOn(boolean)		Chooses appropriate color based on team
	Class Name	Clue			
	Description	Used to set the styling based on team			
	Attributes	**Visibility**	**Data Type**	**Name**	**Description**
	Methods	**Visibility**	**Method Name**		**Description**
		private	Clue(boolean)		Constructor
	Class Name	Score			
	Description	Used to set the styling based on team			
	Attributes	**Visibility**	**Data Type**	**Name**	**Description**
	Methods	**Visibility**	**Method Name**		**Description**
		private	Score(boolean)		Constructor
	Class Name	SpyImage			
	Description	Used to set the styling based on team			
	Attributes	**Visibility**	**Data Type**	**Name**	**Description**
	Methods	**Visibility**	**Method Name**		**Description**
		private	SpyImage(boolean)		Constructor

4.2.6 class <KeycardPopup>

Class Name	KeycardPopup			
Description	Popup which translates and displays the keycard for the user			
Attributes	**Visibility**	**Data Type**	**Name**	**Description**
Methods	**Visibility**	**Method Name**		**Description**
	public	KeycardPopup(Keycard)		Constructor
Inner Classes	**Class Name**	PopupTitle		
	Description	Styling class for title element		
	Attributes	**Visibility**	**Data Type**	**Name**	**Description**
	Methods	**Visibility**	**Method Name**		**Description**
		private	PopupTitle()		Constructor
	Class Name	KeycardPane			
	Description	Grid which displays the keycard cardtypes.			
	Attributes	**Visibility**	**Data Type**	**Name**	**Description**
	Methods	**Visibility**	**Method Name**		**Description**
		public	KeycardPane(CardType[])		Constructor
		private	getCardColor(CardType)		Translates card type to color
	Inner Class	**Class Name**	KeycardElement		
		Description			
		Attributes	**Visibility**	**Data Type**	**Name**	**Description**
		Methods	**Visibility**	**Method Name**		**Description**
			public	KeycardElement(Color)		Constructor
	Class Name	StartLabel			
	Description	Label to show the score			
	Attributes	**Visibility**	**Data Type**	**Name**	**Description**
	Methods	**Visibility**	**Method Name**		**Description**
		private	StartLabel(boolean)		Constructor
	Class Name	CloseButton			
	Description	Button with closing the keycard window as its only function			
	Attributes	**Visibility**	**Data Type**	**Name**	**Description**
	Methods	**Visibility**	**Method Name**		**Description**
		public	CloseButton()		Constructor
		private	defaultStyle()		Style helper method
		private	pressedStyle()		Style helper method
		private	enteredStyle()		Style helper method
	Inner Classes	**Class Name**	**Description**		**Method**
		CloseHandler	Closes the popup window		handle(ActionEvent)
		EnterHandler	Handles mouse entered events		handle(ActionEvent)
		ExitHandler	Handles the mouse exit event		handle(ActionEvent)
		PressHandler	Handles mouse press events		handle(ActionEvent)
		ReleaseHandler	Handles the mouse release event		handle(ActionEvent)

4.2.7 class <Style>

Class Name	Style		
Description	An interface with all the styling constants for the program		
Constants	**Data Type**	**Name**	**Description**
	double	WINDOW_SIZE_WIDTH	
	double	WINDOW_SIZE_HEIGHT	
	Color	WINDOW_COLOR_BACKGROUND_DARK	
	Color	WINDOW_COLOR_BACKGROUND_LIGHT	
	Color	WINDOW_COLOR_BORDER	
	Color	WINDOW_COLOR_TEXT_LIGHT	
	Color	WINDOW_COLOR_TEXT_DARK	
	Color	WINDOW_COLOR_CARD_BLACK	
	Color	WINDOW_COLOR_CARD_YELLOW	
	Color	WINDOW_COLOR_CARD_RED	
	Color	WINDOW_COLOR_CARD_BLUE	
	Color	WINDOW_COLOR_CARD_HIDDEN	
	int	WINDOW_SIZE_DEFAULT	
	int	WINDOW_SIZE_HEADER	
	String	WINDOW_FONT_FAMILY	
	FontWeight	WINDOW_FONT_WEIGHT	
	Font	WINDOW_FONT_DEFAULT	
	Font	WINDOW_FONT_HEADER	
	double	CARD_WIDTH	
	double	CARD_HEIGHT	
	CornerRadii	CARD_CORNER	
	Border	CARD_BORDER_DEFAULT	
	Border	CARD_BORDER_ENTERED	
	Background	CARD_BACKGROUND_HIDDEN	
	Background	CARD_BACKGROUND_YELLOW	
	Background	CARD_BACKGROUND_BLACK	
	Background	CARD_BACKGROUND_BLUE	
	Background	CARD_BACKGROUND_RED	
	Border	HQ_BORDER	
	Background	HQ_BACKGROUND_RED_DEFAULT	
	Background	HQ_BACKGROUND_RED_ON	
	Background	HQ_BACKGROUND_BLUE_DEFAULT	
	Background	HQ_BACKGROUND_BLUE_ON	
	Insets	HQ_TEAM_PADDING	
	Pos	HQ_TEAM_ALIGNMENT	
	int	HQ_CLUE_HEIGHT	
	Insets	HQ_CLUE_PADDING	
	Pos	HQ_CLUE_ALIGNMENT	
	CornerRadii	HQ_CLUE_CORNER_LEFT	
	CornerRadii	HQ_CLUE_CORNER_RIGHT	
	Background	HQ_CLUE_BACKGROUND_LEFT	
	Background	HQ_CLUE_BACKGROUND_RIGHT	
	int	HQ_SCORE_WIDTH	
	int	HQ_SCORE_HEIGHT	
	Pos	HQ_SCORE_ALIGNMENT	
	Insets	HQ_SCORE_MARGIN	
	Border	HQ_SCORE_BORDER	
	Background	HQ_SCORE_BACKGROUND_LEFT	
	Background	HQ_SCORE_BACKGROUND_RIGHT	

	int	HQ_IMAGE_WIDTH	
	int	HQ_IMAGE_HEIGHT	
	Image	HQ_IMAGE_LEFT	
	Image	HQ_IMAGE_RIGHT	
	double	FIELD_GAP_H	
	double	FIELD_GAP_V	
	Pos	FIELD_ALIGNMENT	
	Insets	FIELD_PADDING	
	Background	FIELD_BACKGROUND	
	int	CONTROL_HEIGHT	
	Background	CONTROL_BACKGROUND	
	Border	CONTROL_BORDER	
	int	CONTROL_ELEMENT_HEIGHT	
	int	CONTROL_ELEMENT_WIDTH	
	Insets	CONTROL_ELEMENT_MARGIN	
	Border	CONTROL_ELEMENT_BORDER_DEFAULT	
	Border	CONTROL_ELEMENT_BORDER_ENTERED	
	Background	CONTROL_ELEMENT_BACKGROUND_DEFAULT	
	Background	CONTROL_ELEMENT_BACKGROUND_PRESSED	
	Insets	POPUP_PADDING	
	CornerRadii	POPUP_CORNER	
	Background	POPUP_BACKGROUND	
	Border	POPUP_BORDER	
	Insets	POPUP_TITLE_PADDING	
	Insets	POPUP_TITLE_MARGIN	
	Background	POPUP_TITLE_BACKGROUND	
	double	POPUP_GRID_GAP_H	
	double	POPUP_GRID_GAP_V	
	double	POPUP_GRID_GAP_V	
	Insets	POPUP_GRID_PADDING	
	Insets	POPUP_GRID_MARGIN	
	Border	POPUP_GRID_BORDER	
	int	POPUP_GRID_ELEMENT_CORNER	
	int	POPUP_GRID_ELEMENT_HEIGHT	
	int	POPUP_GRID_ELEMENT_WIDTH	
	Insets	POPUP_START_PADDING	
	Insets	POPUP_START_MARGIN	
	Background	POPUP_START_BACKGROUND_BLUE	
	Background	POPUP_START_BACKGROUND_RED	
	Insets	POPUP_CLOSE_MARGIN	
	int	POPUP_CLOSE_HEIGHT	
	int	POPUP_CLOSE_WIDTH	
	Border	POPUP_CLOSE_BORDER_DEFAULT	
	Border	POPUP_CLOSE_BORDER_ENTERED	
	Background	POPUP_CLOSE_BACKGROUND_DEFAULT	
	Background	POPUP_CLOSE_BACKGROUND_PRESSED	

4.3 Controller

The controller accepts the user's input and calls the model and view to complete the user's needs. The controller itself does not output anything and does any processing. It simply receives the request and decides which model component to call to process the request, and then determines which view to use to display the returned data.

4.3.1 class <Controller>

Class name	Controller			
Description	Controller object that handles the processing of Messages from the View (GUI)			
Attributes	**Visibility**	**Data type**	**Name**	**Description**
	Private	Stack<Message>	mMessageStack	Stores the message from view
	Private	Stack<Message>	mUndostack	Stores the undo message from view
	Private	ArrayList<Message>	mMessageLog	Stores the message logs
	Private	Keycard[ ]	mKeyCardCollection	Stores all key cards.
	Private	GameBoard	mGameBoard	Stores 5*5 game board object.
	Private	Inbox	mInbox	Stores inbox message
	Private	Outbox	mOutbox	Stores outbox message
	Private	Scene	mGameScene	Stores game scene
Methods	**Visibility**	**Method Name**	**Description**	
	Public	Keycard[ ]	Constructor	
	Public	Inbox	Constructor	
	Public	Outbox	Constructor	
	Public	setGameScene	Setter	
	Public	KeyCard getKeyCard	Getters	
	Public	ArrayList<Card> getCardList	Getters	
	Public	ArrayList<Message> getMessageLog	Getters	
	Private	selectKeyCard	Random select keycard	
	Private	startNewGame	Create new game	
	Public	update	Update the game informations	

4.3.2 class <GameOberver>

Class Name	GameObserver		
Description	Game Observer object that handles card updates and notifies changes to the View (GUI)		
Attributes	**Visibility**	**Data type**	**Name**
	Private	Outbox	mOutbox
	Private	Int	mBlueScore
	Private	Int	mRedScore
	Private	Boolean	mBlueTurn
	Private	Stack<Boolean>	mTurnStack
Methods	**Visibility**	**Method Name**	**Description**
	Public	KeyCard	Constructors
	Public	Outbox	Constructors
	Private	initializeGame	initializing the game
	Private	incrementBlue	Blue team added one point
	Private	incrementRed	Red team added one point
	Private	decrementBlue	Blue team reduced by one point
	Private	decrementRed	Red team reduced by one point
	Private	sameTurn	Red or Blue team will continue the game
	Private	nextTurn	This team of games ends, let the other team continue the game
	Private	previousTurn	Indicates that the last player was that Team
	Public	Update	Update the game informations

4.3.3 class <Inbox>

Class Name	Inbox		
Description	Inbox object that receives messages from the View (GUI) and notifies the Controller.java object		
Attributes	**Visibility**	**Data type**	**Name**
	Private	Message	mMessage
Methods	**Visibility**	**Method Name**	**Description**
	Public	Inbox	Constructors
	Public	Message getMessage	Getters
	Public	sendMessage	Update the message and notify observer

4.3.4 class <Message>

Class Name	Message		
Description	Message object used to send information to the control section from the view section		
Attributes	**Visibility**	**Data type**	**Name**
	Private	MessageType	mType
	Private	Int	mCardAffected
Methods	**Visibility**	**Method Name**	**Description**
	Public	Message	Constructor
	Public	MessageType	Constructor
	Public	Int	Constructor
	Public	MessageType getMessageType	Getters
	Public	Int getCardAffected	Getters

4.3.5 class <MessageType>

Class Name	MessageType			
Description	MessageType enum used to define the types of messages sent from the View to Control			
Attributes	**Visibility**	**Data type**	**Name**	**Description**
	Public	Enum	MessageType	NEW_GAME, NEXT, UNDO, REDO, SELECT

4.3.6 class <Outbox>

Class Name	Outbox		
Description	Outbox object that receives replies from the GameObserver.java object and notifies the View (GUI)		
Attributes	**Visibility**	**Data type**	**Name**
	Private	Reply	mReply
Methods	**Visibility**	**Method Name**	**Description**
	Public	Outbox	Constructors
	Public	Reply getReply	Getters
	Public	sendReply	Update the Reply and notify observer

4.3.7 class <Reply>

Class Name	Reply		
Description	Reply object used to send information from the control section to the view section		
Attributes	**Visibility**	**Data type**	**Name**
	Private	ReplyType	mType
	Private	Int	mCardAffected
	Private	CardType	mCardType
	Private	Int	mBlueScore
	Private	Int	mRedScore
	Private	boolean	mBlueTurn
Methods	**Visibility**	**Method Name**	**Description**
	Public	ReplyType	Constructor
	Public	Int	Constructor
	Public	CardType	Constructor
	Public	Int	Constructor
	Public	Int	Constructor
	Public	boolean	Constructor
	Public	ReplyType getReplyType	Getters
	Public	Int getCardAffected	Getters
	Public	CardType getCardType	Getters
	Public	int getBlueScore	Getters
	Public	Int getRedScore	Getters
	Public	boolean getCurrentTurn	Getters

4.3.8 class <ReplyType>

Class Name	ReplyType
Description	Reply enum used to define the types of replies sent from Control to View
Visibility	Public
Data type	Enum
Name	ReplyType

4.3.9 class <Strategy>

Class Name	Strategy			
Description	Strategy class (static) with methods to process a Computer turn			
Attributes	**Visibility**	**Data type**	**Name**	**Description**
	Public	Int	pickRandomCard(ArrayList<Card>)	Random pick the card
	Public	Int	pickNextCard(ArrayList<Card>)	selected the next card and pick it up

5 Glossary

Agents
One of the potential outcomes of a codename card when revealed. Agents will be of either Red or Blue type. Revealing an agent of the other team is like scoring a point; if all agents of the other team are revealed, you win.

Assassin
One of the potential outcomes of a codename card when revealed. The team that reveals the assassin immediately loses the game.

BoardGameGeek
A well-known website that databases thousands of board and card games that hosts awards known as Golden Geek awards.

Bystander
One of the potential outcomes of a codename card when revealed. It represents a "neutral" card that advances neither team. Revealing an innocent Bystander ends your turn.

Codenames
Codenames refers to the name of the board game as a whole, as well as to individual "code names" that are the words on cards to be guessed.

Eclipse
An open source integrated development environment (IDE) compatible with Java that is the listed workspace requirement for the project.

Golden Geek
A prestigious award given to board and card games by BoardGameGeek.

Hint
A single word and number given by the Spymaster to the Operative on their turn before the Operative chooses a card to reveal.

JUnit
A tool for developers that allows for testing on the Java virtual machine (JVM). It is, much like Eclipse, one of the tools used by the team to create this game.

Keycard
A card with a diagram on it given to the Spymasters of each team so that the Spymasters know which cards are agents of each team and which card is the Assassin.

MVC
An commonly-used programming architectural pattern for software developers consisting of three parts: the Model, the View, and the Controller.

Operative
One of the two roles of player for the game Codenames. There is one Operative on each team, and the Operative is in charge of revealing a card (in the form of a guess) after the Spymaster has given them a hint.

Spiel des Jahres
A prestigious award given to board and card games. The term is German and means, quite literally, "Game of the Year".

Spymaster
One of the two roles of player for the game Codenames. There is one Spymaster on each team, and the Spymaster is in charge of giving the Operative on their team a hint, knowing the layout of the Agents, Assassin and Bystanders while the Operative does not.

UML
Abbreviation for Unified Modeling Language. It is a standardized way of laying out the diagram of a model in software engineering.

User
The person using the program Codenames. The User is not necessarily the Operative, Spymaster or even playing the game; they are simply using the program.

Word
Every codenames card is associated with a word. The Operative must choose a word when guessing, based upon the hint given by the Spymaster.

6 References

Codenames, How to play Codenames, YouTube
`https://www.youtube.com/watch?v=J8RWBooJivg` (current February 7th 2019)

A Description of File Format: Tasks

Member Name	Task
Annes Cherid	Organizer
David Boivin	Coder
Karim Loulou	Coder
Kevin McAllister	Coder
Yogesh Nimbhorkar	Coder
Souheil Al-Awar	Documentor
Benson Chan	Documentor
Carl Neil Cortes	Documentor
Gaoshuo Cui	Documentor
Robert Laviolette	Documentor
Ke ma	Documentor

B Description of File Format: Persons

B.1 Kevin McAllister (40031326)

Date	Task	Duration
Jan. 10	Created GitHub repository and Slack Group	1 Hour
Jan. 16	GROUP MEETING - Added members to Slack and GitHub and decided on general plan for the project	2 Hours
Jan. 18	Meeting with Karim - Brainstormed game structure and requirements and wrote simple objects for the game	3 Hours
Jan. 23	GROUP MEETING - Discussed game structure and organized a clear idea of how to write the code	2 Hours
Jan. 24	Updated objects to match discussed model and set up communication between View and Control	2 Hours
Jan. 25	Added more objects to store data and updated code with proper names and necessary TODOs	2.5 Hours
Jan. 27	Substantial code changes, functionality changes and revision on data storage and enum / variable naming	4 Hours
Jan. 31	GROUP MEETING - Went over Demo requirements and discussed further on the code communication	2 Hours
Feb. 1	Reworked the Observer communication system and cleaned the code	1.5 Hours
Feb. 2	Added new logic class to process game updated and notify the View	4 Hours
Feb. 3	Added Strategy class with Operative Random / Next strategies, code bugfixes	1 Hour
Feb. 4	Meeting with Karim - Discussed methods to implement new game and word loading, general discussion about project structure	1 Hour
Feb. 5	Added new game functionality, more bug fixes, implemented word loading	2 Hour
Feb. 6	Set up JUnit testing to test all cases discussed previously (and more) polished parts of the game	5 Hours
Feb .7	More cleanup on code, added ability to choose Operative strategy types	2 Hours
	Total	35 Hours

B.2 Benson Chan (40046280)

Date	Task	Duration
Jan. 18	Made Github and Slack account	0.5 Hours
Jan. 23	Group meeting work division	1.75 Hours
Jan. 31	Revision of group meeting and revised Google docs	1.5 Hours
Feb. 2	Looked at use case document	0.5 Hours
Feb. 5	Group meeting predemo + worked on use case diagram	1.75 hours
Feb. 6	Finish use case diagram + table	1 Hour
	Total	7 Hours

B.3 Annes Cherid (40038453)

Date	Task	Duration
Jan. 16	Full Group Meeting 1	1.5 Hours
Jan. 16	Creating and setting up GitHub/Slack	0.5 Hours
Jan. 17	Individual work - Documenting the meeting	1 Hour
Jan 18.	Coders meeting	1 Hour
Jan. 22	Documentors group meeting 1	1 Hour
Jan. 23	Individual work - Documenting the meeting	1.5 Hours
Jan. 23	Full Group meeting 2	1.75 Hours
Jan. 24	Individualwork and Documenting the meeting	1 Hour
Jan. 30	Full Group meeting 3	1.75 Hours
Jan. 31	Individual work - Documenting the meeting	2 Hours
Feb. 5	Work on personal diary	1 Hour
Feb. 5	Documentors group meeting 2	1 Hour
Feb. 5	INDIVIDUAL WORK and Documenting the meeting	1 Hour
Feb. 6	Full group meeting 4	1.5 Hours
Feb. 6	Work on personal diary	1 Hour
Feb. 6	Work on full team diary(+LaTeX)	2 Hours
Feb. 7	Misc work	0.5 Hours
Feb. 7	Work on full team diary(+LaTeX)	1 Hour
	Total	22 Hours

B.4 David Boivin (40004941)

Date	Task	Duration
Jan. 16	Attended first team meeting and connecting accounts on required platform	2 Hours
Jan. 17	Worked on basic MVC template , especially the View section	0.5 Hours
Jan.18	Drew up GUI hierarchy	0.5 Hours
Jan. 19	Worked locally on layout design template	2 Hours
Jan. 23	Attended second team meeting	2 Hours
Jan. 29	Finished laypout design and uploaded to GitHub	4 Hours
Jan. 30	Attended third team meeting	2 Hours
Jan. 31 - Feb. 1	Finalized GUI styling and made necessary changes to layout	10 Hours
Feb. 5	Implemented linking mechanism between View and Controller	1 Hour
Feb. 6	Added end game Popup and bug fixing	2 Hours
Feb. 7	Attended fourth team meeting	2 Hours
Feb. 8	Made tweeks and refactored View	6 Hours
	Total	34 Hours

B.5 Ke Ma (26701531)

Date	Task	Duration
Jan. 16	Group Meeting	2 Hours
Jan. 20	Created GitHub and Slack account	0.5 Hours
Jan. 22	Documentors meeting - work division for first half	1.5 Hours
Jan. 23	Worked on document's 'purpose' part	2 Hours
Jan. 30	Group meeting - work division for second half	2 Hours
Feb. 3	Individual - MVC Architecture	3 Hours
Feb. 5	Individual - MVC Architecture	3 Hours
Feb. 6	Individual - LaTeX document creation	3 Hours
	Total	17 Hours

B.6 Souheil Al-awar (26558038)

Date	Task	Duration
Jan. 11	Watched a 1 hour video to learn LaTeX, 30 minutes video to learn the game and took notes	1.5 Hours
Jan. 16	LAB - Got familiarized with Github, Slack, Codenames game structure, Latex and met my team.	2 Hours
Jan. 22	GROUP MEETING - Met new team members, answered questions, and gave each other tasks.	1 Hour
Jan. 23	LAB - Divided the documentation outline and talked about what everything in the outline is.	1.5 Hours
Jan. 24	Asked questions to the tutor about the domain model and started reading the slides until midnight.	2 Hours
Jan. 25	Made a list with the possible domain elements with its relationships and made my first draft.	2.5 Hours
Jan. 30	LAB - Started designing the domain model in my laptop and asked questions to the TA.	2.5 Hours
Feb. 2	Confirmed the domain elements with my team and desgined the first domain model.	4 Hours
Feb. 5	GROUP MEETING - Got ready to demo, did some touches to the LaTeX doc and domain model.	2 Hours
Feb. 6	LAB - Did not demo, started working on a domain model with attributes.	1.5 Hours
Feb. 7	Finished the second domain model, made intro for both and showed work to my team.	3.5 Hours
	Total	24 Hours

B.7 Gaoshuo Cui (40085020)

Date	Task	Duration
Jan. 16	GROUP MEETING1-Meet everyone and build connections, and clarify our own tasks	2 Hours
Jan. 17	Understand the rules of the game, learn about the entire first phase of the project	1 Hours
Jan. 19	Learn to use latex	2 Hour
Jan. 22	GROUP MEETING2-understand of domain modeling and use cases	1 Hour
Jan. 23	GROUP MEETING3-Assign the contents of the requirement document	2 Hours
Jan. 25	Learning domain model and use case	2 Hours
Jan. 29	Complete the context part and write the rules of the game in detail	2 Hours
Jan. 30	GROUP MEETING4-Discussion of use case and domian model	2 Hours
Feb. 5	Complete the controller part of the mvc and spymaster in the use case	3 Hours
Feb. 6	GROUP MEETING5- Did not demo, started working on LaTex	1.5 Hours
	Total	18.5 Hours

B.8 Carl Neil Cortes (40016567)

Date	Task	Duration
Jan. 16	Group Meeting - started organization for the project.	2 Hours
Jan. 22	Documentor Meeting - Documentors were given further instructions for iteration 1. Robert joins the team.	1.5 Hours
Jan. 23	Looked up how to create a LaTeX file and created prototype	1.5 Hours
Jan. 23	Group Meeting - showed prototype of LaTeX file on laptop and divided the documentor's tasks amongst ourselves	2 Hours
Jan. 24	Did research on use cases and uploaded a rough draft	1 Hour
Jan. 25	Posted useful chapter on domain models on Slack channel	0.5 Hours
Jan. 29	Added all of the documentors parts onto LaTeX document	1 Hour
Jan. 31	Group Meeting - I presented LaTeX file to the t.a., further tasks were distributed amongst the documentors	2 Hours
Feb. 5	Group meeting - game presentation to documentors, documentors finish their use cases.	1 Hour
Feb. 8	Started to add all of the required information for the first iteration onto the LaTeX file.	2 Hours
Feb. 9	Added missing information onto the LaTeX document	5 Hours
	Total	19.5 Hours

B.9 Robert Laviolette (27646666)

Date	Task	Duration
Jan. 22	Documenters First Meeting	1 Hour
Jan. 22	LaTeX Familiarity	0.5 Hours
Jan. 23	Second Group Meeting	2 Hours
Jan. 23	Document Work	0.25 Hours
Jan. 30	Third Group Meeting	2 Hours
Feb. 4	MVC Work	1.5 Hours
Feb. 5	Fourth Group Meeting	1 Hour
Feb. 5	MVC - View Work	1.5 Hours
Feb. 6	Fifth Group Meeting	1.5 Hours
Feb. 7	Document Quality Control	5 Hours
Feb. 8	Glossary Work	1 Hour
	Total	17.25 Hours

B.10 Karim Loulou (40027203)

Date	Task	Duration
Jan. 16	GROUP MEETING - Added members to Slack and GitHub and decided on general plan for the project	2 Hours
Jan. 17	First draft of the design	2 Hours
Jan. 18	Meeting with Kevin	1 Hour
Jan. 23.	Team Meeting	2 Hours
Jan. 25	Code part of the model	3 Hours
Jan. 29	Create the database	2 Hours
Jan. 30	GROUP MEETING - Went over Demo requirements and discussed further on the code communication	2 Hours
Feb. 4	Meeting with Kevin to debug	1 Hour
Feb. 5	Testing part of the model	4 Hours
	Total	19 Hours