Group: Group 8

Group name: Ace and Speed Masters

GitHub : <https://github.com/g24e2316/Group-8/edit/main/README.md>

Ekasi Jack:

Lelethu Conjwayo & Caitlin Elliott – both contributed equally in all aspects of the game

Speed Masters :

Mogau Seroka – Coding and game development

Mihle Ndabula – Design, testing phases, improvement of game functionality

**Background and Motivation**

Ekasi Jack is a friendlier version of the Black jack card game. The original Black Jack game was invented during the 1700’s by the French/ Italian and it is a gambling game that was invented during World War I (Rainbow Casino, 2020). Ekasi Jack doesn’t have an exact background history because it is a game that one of our team members grew up playing and was passed down by generations before her.

The primary motivation and aim for Ekasi Jack is to create a friendlier and more laid - back version of black jack. There are many people who do not like to be mentally stimulated while playing games. They want to play games to unwind without too many heavy game rules. Ekasi Jack provides a more relaxing and entertaining environment for such individuals.

Speed Masters was created to provide an adrenaline rushing and challenging experience for people who enjoy arcade style racing games. The aim of the game is to combine entertainment with skill development, like hand- eye coordination and thinking strategically. The game creates a fun loop that encourages individuals to continue playing the game.

**Problem statement**

A lot of games do not easily encourage players to play because they are complex and too difficult to learn. There needs to be a game that balances challenging with appealing to both casual and competitive players. Speed Masters mitigates this gap by giving straightforward controls along with increasingly difficult gameplay. This makes sure that players stay engaged while developing the skills mentioned above.

Similarly to Speed Masters, a lot of games are too complex to understand. They require focus and Ekasi - Jack accommodates that aspect by simplifying and making the game more understandable and enjoyable for everyone.

**Approach**

Speed Masters:

How:

* research: looking at the existing racing games like roadblusters to find successful elements.
* design: creating a simple user interface and gameplay mechanics
* development: we used Java’s AWT and swing libraries for graphics and handling events.
* Testing: we conducted multiple rounds of playtesting so we can refine gameplay accordingly.

Milestones:

* Developing the concept: final game design
* Prototype creation: developing an initial playable version
* Testing: getting feedback from small groups of testers
* Final release: published the game

Ekasi Jack:

As mentioned in our project proposal we have maintained storing the card and deck classes in array lists.

The game remains multiplier and we have decided to not make a single player version of the game. The point of the game is to be played with various people to make it more fun which is why a multiplayer format suits the game’s vision well.

Unfortunately because our game is only functional on terminal the players will be able to see each other’s cards, which is a limitation that we acknowledge. The game has not been implemented on the graphical user interface as planned in the proposal so we did not use any applications on making it more visually appealing.

Initially we had point calculators on determining winners but we made changes to simplify the game not only for the users, but for the extent of our programming ability as well. The focus of the game is to determine the loser rather than the winners. The game is mostly focused on the loser who is the person that will be left with the Jack of spades card at the end of the game.

**SDLC**

Speed masters:

1. Requirement analysis: finding key features and player needs.
2. Planning: Setting timelines and resource to be used
3. Design: In the game, the player can move left and right, pause the game, resume it, restart it and quit. There are squares moving in opposite direction to that of the player and if the player hits it the game ends.
4. Implementation: writing code using Java's AWT and Swing libraries.
5. Testing: different types of testing to see potential threats that could break the games code or disrupt the game.
6. Deployment: submitting the game (ask group members)
7. Maintenance: Providing updates based on user feedback.

An agile model was used to allow iterative development and flexibility in response to the players (testers) feedback throughout the project life cycle.

Ekasi Jack:

Our game follows the Waterfall Model framework because the game requirements were well defined from the beginning. Before we moved to another phase we reviewed and decided which changes to make before moving on.

Ekasi-jack is straightforward and it has a well-defined scope, the model has allowed us to breakdown the game into smaller manageable phases and to complete each one before moving on to the next.

There were two of us working on the game and this helped us understand what we had to focus on and which responsibilities each of us had. For example, in the deck class we knew we had to have a deck that represents card objects that has to be shuffled. It also had to allow us to remove cards like we would in real life. The next part we tackled was the player class etc.

Visuals

Speed Masters

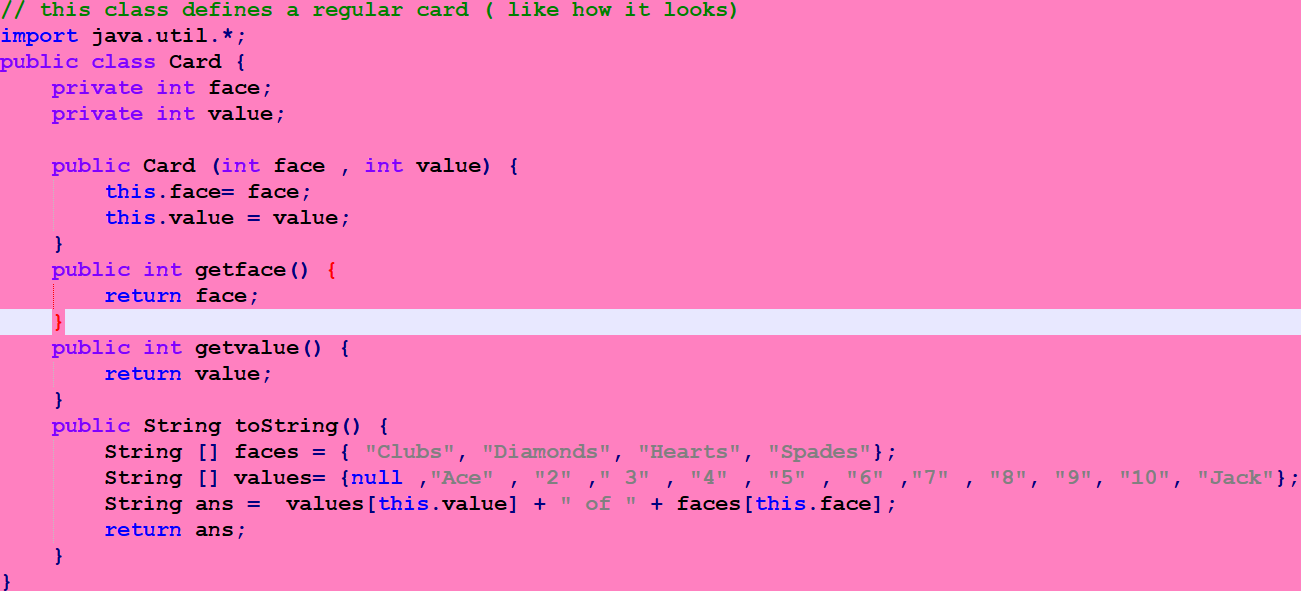
|  |
| --- |
| GAME |
|  |
|  |
| + main():void |

|  |
| --- |
| work |
| -space:int  -speed:int  -width:int  -height:int  -WIDTH:int  -HEIGHT:int  -move:int  -count:int  -countPrev:int  -ocars:ArrayList<Rectangle>  -rand:Random  -linef:boolean  -isPaused:boolean  -isGameOver:boolean  -t:Timer |
| work()  -initialize():void  +addocars(boolean first):void  +paintComponent(Graphics g):void  +actionPerformed(ActionEvent e):void  +moveLeft():void  +moveRight():void  +keyPressed(KeyEvent e):void  +keyReleased(KeyEvent e):void  +keyTyped(KeyEvent e):void |

Ekasi Jack

The card class

* The card class provides a representation of a card object
* The card object has two attributes, the face(suit) and the value(rank)



The deck class

* This class represents the deck of cards that the game will be using
* It allows for shuffling
* Distributing cards to the players]

A screenshot of a computer code

Description automatically generated

The player class

* This class represents the ‘players’ component of the game
* It controls the management of the player names
* Allows the players to interact with each other

A screenshot of a computer program

Description automatically generated

Hand class

* The hand class essentially deals with the cards in each players hand
* The swap method in the hand class is used to pass cards from one player to the next

A black text on a white background

Description automatically generated

Game flow class

* This is the main logic of the game
* Takes user input to assess how many players are in the game this is important because it affects the way cards are distributed
* We used an array to store player objects because it allows us to manage multiple instances of the class easily
* By doing that it made it easy to access and manipulate individual player

A screenshot of a computer code

Description automatically generated

Ekasi Jack UML:

|  |
| --- |
| Card |
| -face : int  - value: int |
| + Card(face: int , value : int):void +getface(): int  +getvalue():int  +toString():String |

|  |
| --- |
| Deck |
| -cards:ArrayList<Card> |
| + Deck():void  +getCards():ArrayList<Card>  + Distribute(n int) : List<Card>  +getsize():int  +toString:String |

|  |
| --- |
| Program |
| +main(args:String [ ] ):void |

|  |
| --- |
| Player |
| -name :String  -hand:ArrayList<Card> |
| +Player(name:String)  +getName():String  +getHand():ArrayList<Card> +addcards(card:Card)  +removecards(card Card)  +getSize():int  +matchingcards():void  +passcards(anotherplayer : Player)void  +Loser (): Boolean  +nocards():Boolean |

|  |
| --- |
| Hand |
| -hand :ArrayList<Card> -changingCard: Card  -name: String  -handSize: int  -player: Player |
| +Hand(hand: ArrayList<Card>)  +add (card Card): void  +Remove(card Card): void  +swap(card Card): void  +isEmpty(): Boolean  +getHandSize(): int  +getName(): String  +toString(): String |

|  |
| --- |
| GameFlow |
| -deck:Deck  -players:Player[]  -numplayers:int |
| +GameFlow()  +playersMethod():void  +getPlayers():Player[ ]  +handingCardMethod():void  +Play():void |