

Table of Contents

Card (Card.h, Card.cpp):.....	2
Methods:.....	2
Variables:	2
Deck (Deck.h, Deck.cpp):	3
Methods:.....	3
Variables:	4
Game (Game.h, Game.cpp):	5
Methods:.....	5
Variables:	6
GameButton (GameButton.h, GameButton.cpp):	8
Methods:.....	8
Variables:	9
Person (Person.h, Person.cpp):.....	10
Methods:.....	10
Variables:	Error! Bookmark not defined.
Player (Player.h, Player.cpp – Derived from Person):.....	11
Methods:.....	11
Dealer (Dealer.h, Dealer.cpp – Derived from Person):	11
Methods:.....	11
Resources (Resources.h, Resources.cpp):.....	11
Methods:.....	11
Variables:	12
cardFormation (Resources.h):	13
Methods:.....	13
Variables:	13
enum E_personType:	13
states:.....	13
External documentation:	13

Card (Card.h, Card.cpp):

Methods:

Constructor – *Card(const int &CardID)*

This overloaded version of the constructor takes in a card ID, which is a value from 0 to 51. This card ID controls what card is created, the cards are in order, and in groups of 13 (13 cards in each suit).

Copy Constructor – *Card(Card &OtherCard)*

Used the assignment operator to copy from another card, does a deep copy of all of the other card's members.

getSprite() – sf::Sprite& getSprite()

Returns the sprite assigned to the card. Each card has a different sprite (not to be confused with a texture, as each sprite holds a pointer to a texture, which means two identical cards will have two different sprites which point to the same texture)

generateCard – private function– void generateCard(const int &cardID)

Generates the name of a card, and assigns the correct texture to its sprite. Is private, so it is only to be used within the card class.

getCardId – int getCardId()

Returns the ID of the card (a number from 0 to 51, the range of cards in a standard deck without jokers)

getCardNum – int getCardNum()

Returns the Number of the card relative to its suit (0-12). Cards 2 to 9 will return their number minus 1. Ace is 0, Jack is 10, Queen is 11 and king is 12

*getName – const char * getName()*

Returns the name of the card, for example: "King of Hearts"
This is the name that is generated in generateCard()

Assignment Operator Override – void operator = (Card &Ocard)

Does a deep copy of the card class.
Copies its sprite, but does not copy its texture.

Destructor – ~Card()

Deletes the name of the card.

Variables:

m_cardID – int m_cardID

Id of the card (0 to 51, can be any card in a standard deck without jokers)

m_cardNum – int m_cardNum

The number of the card (0 to 12, the amount of cards in each suit)
0 = Ace, 1-9 = 2 to 10, 10 = Jack, 11 = Queen, 12 = King

m_cardSuit – int m_cardSuit

The suit of the card (0 to 3, each number being a different suit)

*m_myName – char * m_myName*

The name of the card which is generated by the “generateCard()” function.
E.g “King of Hearts”

m_mySprite – sf :: Sprite m_mySprite

An instance of the card’s sprite which points to a texture in the game resources.

Deck (Deck.h, Deck.cpp):

Methods:

Constructor – *Deck()*

Sets up the deck

shuffle – void shuffle()

Shuffles the deck by creating a copy of the deck and copying random elements from it back to the original deck.

generateMainDeck – void generateMainDeck()

Generates the main deck. This deck holds all 52 cards and gets shuffled. This function also makes sure that the card texture has been rendered in the resources class.

clearDeck – void clearDeck()

Clears the deck and deletes every card.

drawDeck – void drawDeck(int x, int y, int separation)

This function uses the window context in the game class to draw the cards on the screen. These cards are drawn at a position (x,y) and are separated by the separation parameter.

addCard – void addCard(int cardID)

Creates a new card in the deck, given the CardID (value from 0 to 51)

getWidth – int getWidth(int separation)

Returns the drawn width of the deck given a separation. This is helpful when centering the deck.

getHeight – int getHeight()

Gets the draw height of the deck or the drawn height of a single card, as they are the same. Will return 0 if the deck is empty.

getSize – unsigned int getSize()

Gets the size of the deck, this is the actual amount of cards that are in the deck.

calculateTotal – int calculateTotal()

Calculates the total value of a deck, counts all cards aside from aces first, then counts aces to make sure that they are valued at 1 (hard ace) when it would cause the value to be over 21 otherwise.

*getCard – Card * getCard(unsigned int ID) const*

Returns a pointer to a card located at a certain index in the deck.

*Subscript operator overload – Card * operator[](unsigned int ID) const*

Returns a pointer to a card located at a certain index in the deck. Uses the subscript operator to do so instead of function parameters.

*takeFromDeck – Card * takeFromDeck(Deck &otherDeck)*

Takes a card from the top of another deck and adds it to the top of the deck calling the function.

Destructor – ~Deck()

Deletes every card in the deck.

Variables:

*m_myDeck – std :: vector < Card *> m_myDeck*

A vector of cards that the deck owns. These are pointers so that they can easily be transferred to other decks.

m_valueRect – sf :: RectangleShape m_valueRect

A drawable rectangle that shows the value of a deck.

m_totalText – sf :: Text m_totalText

Drawable text that is the value of the deck, this is drawn within the m_valueRect rectangle.

m_cachedTotal – int m_cachedTotal

A cached version of the total value of the deck.

m_totalChanged – bool m_totalChanged

A Boolean which is true when cards are added or removed from the deck. If the deck changes then its value also changes, so if this is changed, the next time the value is needed, it is recalculated and cached.

Game (Game.h, Game.cpp):

Methods:

instance – *static Game& instance()*

Creates an instance of the Game class if one does not exist, then return a reference to this instance.

When the instance is created, the constructor is called.

This pretty much makes the class a singleton.

destroy – *void destroy()*

Deletes the singleton and sets it to nullptr. Will call the destructor on the singleton instance.

Constructor – Private function – *Game()*

Sets up everything necessary for the game. The main deck, some animation variables, some buttons, some sprites, and the main players (dealer and player).

This also calls “setupSymbolPositions()”

Copy Constructor – Private function – *Resources(Resources& otherResources)*

Defines an empty copy constructor in private. This should make the class non copyable, which is what a singleton should be.

Copy Assignment Operator – Private function – *Resources(Resources& otherResources)*

Defines an empty copy assignment operator in private. This should make the class non copyable, which is what a singleton should be.

run – *void run()*

Function that contains the main game loop.

draw – *void draw()*

Draw code that gets called by the main game loop.

update – *void update()*

Update code that gets called by the main game loop.

setupSymbolPositions – *void setupSymbolPositions ()*

This function sets up data for the rendering of the cards 2 to 10. These cards have symbols on their bodies that relate to their card suit. In this function, those symbols

have their positions defined so that the card generator in the resources class can render cards by knowing where their symbols are.

startGame – void startGame()

Resets all the game decks (Player, Dealer and Main Deck)
Shows the message saying that it's "Your turn!"
Sets game to playing so that the menu is not drawn.

endGame – void endGame()

Shows the menu and stops showing buttons that do not belong to the menu.
Still renders the previous game after this is called.

*getMainDeck – Deck * getMainDeck()*

Returns a pointer to the main deck.

getWindow – sf::RenderWindow& getWindow()

Returns a reference to the window context. Can be useful for rendering to the window outside of the game class.

startAnimation – private function –

startAnimation(const sf::texture &cardTex, E_personType whoHit, float flyToX, float flyToY)

Starts a card animation, given a card texture, and who hit, and a position, the card will animate, and it will fly towards the deck of the person who hit.
This animation delays the hitting of a deck until the card animation touches the deck.

startGameMessage – private function –

*startGameMessage(const int &x, const int &y, const char * message)*

Shows a message to the screen that fades in and out. While this message is being shown, the player cannot press any buttons, but cards can still be dealt.

Destructor – ~Deck()

Deletes all pointers in the class, this includes: the game deck, the player and dealer objects, all the game and menu buttons, the menu window/ box rectangle, the animation card, and the face down card sprite.

Variables:

m_window – sf::RenderWindow m_window

A window context which things get drawn to.

*m_menuBox – sf::RectangleShape * m_menuBox*

A black box that holds the menu buttons.

*m_cardBack – sf::Sprite * m_cardBack*

A face down card which gets drawn to the background all the time.

*m_animationCard – sf :: Sprite * m_animtionCard*

An instance of a sprite, this changes its texture when either the dealer or the player presses hit, then it starts its animation.

*m_playerObj – Player * m_playerObj*

An instance of a player, this is a derivative of a person.

*m_dealerObj – Dealer * m_dealerObj*

An instance of a dealer, this is a derivative of a person.

*m_gameDeck – Deck * m_gameDeck*

The main game deck, this is in the game class because other classes need to easily access it to take cards from it.

*m_hitButton – GameButton * m_hitButton*

A button in the game (while playing blackjack) that allows the player to hit.

*m_standButton – GameButton * m_standButton*

A button in the game (while playing blackjack) that allows the player to stand.

*m_quitButton – GameButton * m_quitButton*

A button in the menu that quits the game.

*m_playButton – GameButton * m_playButton*

A button in the menu that starts the game.

m_animationState – int m_animationState

The animation state of the card animation. 0 = not drawn, 1 = expanding, 2 = moving towards deck.

m_animMoveSpeed – float m_animMoveSpeed

The speed of the card movement in its animation.

m_animAccel – float m_animAccel

The acceleration of the card movement in its animation.

m_animExpandSpeed – float m_animExpandSpeed

The speed at which the card animation expands.

m_animToX – float m_animToX

The X position of where the card animation has to fly to.

m_animToY – float m_animToY

The Y position of where the card animation has to fly to.

m_animHitPerson – E_personType m_animHitPerson

A way to remember if the dealer or the player pressed the hit button, it can either be “E_enumPlayer” or “E_enumDealer”

m_gameMessage – sf :: Text m_gameMessage

A drawable string of a message that gets shown to the player. (Animated)

m_gameMessageState – int m_gameMessageState

The current animation state of “m_gameMessage”.
0 means it does not get drawn, 1 means that it fades in, 2 means that it fades out.

m_gameMessageSpeed – int m_gameMessageSpeed

The speed at which the animation of “m_gameMessage” fades in and out.

m_playing – bool m_playing

A Boolean which is true when a match has begun. If this is false, then the main menu is draw, if it is true, then you play blackjack.

*m_thisInstance – static Resources * m_thisInstance*

An instance of the singleton. Static so that it always exists, although this does not mean that the class always exists.

GameButton (GameButton.h, GameButton.cpp):

Methods:

Constructor –

*GameButton(int Game& gameRef, int x, int y, int width, int height, const car * buttonText, int fontSize, const sf :: Color& startColour, const sf :: Color& hoverColour, const sf :: Color& pressColour)*

Sets up game button and all the drawable items attached to it.
Is given a string to be drawn over the button.

step – void step()

Polls input. Keeps track of the mouse buttons.

draw – void draw()

Draws the button and its text to the position it was given in the constructor.

isHover – bool isHover()

Returns true if the mouse is hovering over the button.

isPress – bool isPress()

Returns true if the left mouse button was pressed within the last game step, and the mouse is hovering over the button.

isReleased – bool isReleased()

Returns true if the left mouse button was released within the last game step, and the mouse is hovering over the button.

Variables:

m_x– int m_x

The x position of the button.

m_y– int m_y

The y position of the button.

m_width– int m_width

The width of the drawn button.

m_height– m_height

The height of the drawn button.

m_button– sf :: RectangleShape m_button

A drawable rectangle shape for the button.

m_buttonText– sf :: Text m_buttonText

Drawable text that gets drawn on the button.

m_mBLDown– bool m_mBLDown

A Boolean the keeps track of if the left mouse button is pressed.

m_mBLPressed– bool m_mBLPressed

A Boolean the keeps track of if the left mouse button was pressed, however it only remains true for one step of the program.

m_mBLReleased– bool m_mBLReleased

A Boolean the keeps track of if the left mouse button was released, however it only remains true for one step of the program.

m_startColour– sf :: Color m_startColour

The colour the button gets set to when it is created, and when the mouse is not hovering over it.

m_hoverColour– sf :: Color m_startColour

The colour the button gets set to when the mouse is hovering over it, but the left mouse button is not pressed.

m_pressColour – sf :: Color m_startColour

The colour the button gets set to when the mouse is hovering over it and the left mouse button is pressed.

Person (Person.h, Person.cpp):

Methods:

Constructor – *Person()*

Creates a deck for the person, and sets their variables to default.

stand – void stand()

Sets the person as standing.

reset – void reset()

Clears the person's deck and resets their variables to default.

isBust – bool isBust()

Returns true if the value of the person's deck is over 21.

isStanding – bool isStanding()

Returns true if the person is set to standing.

*getDeck – Deck * getDeck()*

Returns a pointer to the deck owned by the person.

*hit – Card * hit()*

Hits, this takes a card from the main deck and places it in the person's deck.

Destructor – *~Person()*

Deletes the person's deck.

*m_myDeck – Deck * m_myDeck*

The deck instance that the person owns. This is actually called a hand, but works in a similar way to a deck.

m_isStanding – bool m_isStanding

Is true if the person instance has used its "stand()" function. Represents if the person is standing in a blackjack game.

Player (Player.h, Player.cpp – Derived from Person):

Methods:

Constructor – `Player()`

Empty, uses the Person constructor.

Dealer (Dealer.h, Dealer.cpp – Derived from Person):

Methods:

Constructor – `Dealer (Game& gameRef)`

Empty, uses the Person constructor.

Resources (Resources.h, Resources.cpp):

Methods:

`loadTexture – void loadTexture(const char * fileName, const char * textureName)`

Loads a texture given a file name and a texture name. The texture name is later used to search through textures.

`loadTexture –`

`void loadTexture(const sf::Texture& copyTexture, const char * textureName)`

Copies a texture and gives it a texture name. The texture name is later used to search through textures.

`renderCard – void renderCard(const int& cardID)`

Using a surface/ render texture, this function creates a card using symbols and images that were loaded in the constructor.

`addToCardFormation – void addToCardFormation (int cardNum, int x, int y)`

Adds a position to a certain card formation, these positions are where symbols will be drawn on cards 2 – 10 (as these cards have symbols on their body)

`copyCardFormation – void copyCardFormation (int sourceNum, int destNum)`

Copies one card formation to another, for example, card number 5 heavily resembles card number 4, so instead of redefining the positions, the positions are copied and the new ones are added.

`destroy – void destroy()`

Deletes the singleton and sets it to nullptr.

`getFont – sf::Font& getFont()`

Returns a reference to the main game font.

*findTexture – sf :: Texture * findTexture(const char * textureName, bool showError = true)*

Searches through textures and checks if their name is the same as the “textureName” argument. Returns a pointer to the texture it finds, or returns nullptr if none are found.

instance – static Resources& instance()

Creates an instance of the Resources class if one does not exist, then return a reference to this instance.

When the instance is created, the constructor is called.

Constructor – Private function – Resources()

Loads resources that will be used by the game (textures and fonts).

Is private so cannot be created from the outside. Gets called by the instance function when the singleton instance is created.

Copy Constructor – Private function – Resources(Resources& otherResources)

Defines an empty copy constructor in private. This should make the class non copyable, which is what a singleton should be.

Copy Assignment Operator – Private function – Resources(Resources& otherResources)

Defines an empty copy assignment operator in private. This should make the class non copyable, which is what a singleton should be.

Destructor – ~Resources()

Deletes all the texture and their associated names.

Variables:

*m_texResources – std::vector < sf::Texture *> m_texResources*

A vector filled with pointers to cards. Holds all the vectors in the game. It holds pointers, as resizing the vector causes the textures to be copied, and their memory address changes, this causes sprites to have their texture pointer to no longer point to the correct texture.

*m_texNames – std::vector < char *> m_texNames*

A vector filled with char * strings, this holds the names of textures that are assigned when a texture is added to the resources. This name is used to find the texture from other classes.

m_cardRenderer– sf :: RenderTexture m_cardRenderer

This is a surface / render texture / drawing buffer which is used to generate cards. Cards components are drawn to this surface then it is copied to a texture to be added to the resources. This surface is reused to save space and time.

`m_cardFont– sf :: Font m_cardFont`

This is the font that is used throughout the main game. It can be accessed using the `getFont()` function.

`m_cardFont– sf :: Font m_cardFont`

This is the font that is used throughout the main game. It can be accessed using the `getFont()` function.

`m_cardFormations– cardFormation m_cardFormations[10]`

An array of card formations. These formations are what the `renderCard()` function uses to create the cards 2-10, as these cards have symbols on their body which need to be drawn at the correct positions.

`m_thisInstance– static Resources * m_thisInstance`

An instance of the singleton. Static so that it always exists, although this does not mean that the class always exists.

cardFormation (Resources.h):

A struct that holds card formations, due to being a struct, its variables don't follow the "m_" member variable naming convention.

Methods:

`addCoords – void addCoords(int Num1, int Num2)`

Adds an x and y coordinate to the "coords" dynamic array.

Variables:

`coords – std::vector < sf::Vector2i > coords`

A dynamic array which holds vectors, these vectors hold x and y positions.

enum E_personType:

An enum that either equals the dealer or the player, this is used for the hit animation, as the animation needs to know who hit by the end so it can add the card to its deck.

states:

`enumPlayer, enumDealer`

External documentation:

All of the classes under the "sf:" namespace are from the SFML library which can all be found here: <http://www.sfm-dev.org/documentation/2.3.2/>

This includes:

- `Sf::Texture`
- `Sf::Sprite`

- Sf::RenderTexture
- Sf::RenderWindow
- Sf::Font
- Sf::Event
- Sf::Vector2f, sf::Vector2i
- Anything starting with sf:: (in the “sf” namespace)

All includes that use the director “SFML” are from the SFML library, for example:

```
#include <SFML/ ... >
```