Project Synopsis - Pakistan Drift

Team Members:
Ayesha Zafar (06113), Muhammad Hasan Khan (07712), Fahad Nadeem (07558)

Project Idea:
"*Pakistan Drift*" is a thrilling car game that immerses players in challenges, requiring lightning-fast reflexes and precision. It takes players into an intense environment where they must skillfully navigate through a never-ending barrage of obstacles on a demanding course. The ultimate objective of the game is to survive these relentless challenges, adding an element of excitement and uncertainty to the experience. Drawing inspiration from the immensely popular "*Traffic Racer*" on the Play Store, our game mimics the excitement of high-speed driving, sharp turns, and formidable obstacles to keep players thoroughly engaged.

Key Features:
- **Welcome Screen:** The game starts with an enticing welcome screen that offers various game options, including selecting difficulty levels, choosing a playing venue (desert, snow, or sunny weather), and initiating gameplay.
- **Game Objects:** "Pakistan Drift" will feature a diverse array of objects, including vehicles, buildings, and stones. Each object will be meticulously designed and incorporated into the game, adding a layer of realism and excitement. Interactive
- **Difficulty Levels**: Players can choose to play at easy, medium, or hard difficulty levels. The speed of the game will increase with each level, intensifying the challenge by accelerating the rate at which obstacles approach.
- **Collision Detection System:** A robust collision detection system will be implemented. When the player's vehicle collides with an obstacle, their health will progressively decrease. If all health bars are depleted, the game will end.
- **Sound Effects**: To enhance the gaming experience, we will include background music and in-game sound effects that react to the gameplay dynamics, making it more immersive.
- **Variety of Venues**: Players can select from different venues from Pakistan such as Skardu Slopes, Thar desert, and Karachi Cityscape, each with its unique visual and environmental characteristics, adding variety and value to the game.

OOP Concepts Implementation:
The design and implementation of "*Pakistan Drift*" will effectively utilize Object-Oriented Programming (OOP) concepts:
- **Encapsulation:** The game's components, such as player, obstacles, health, and scoring, will be encapsulated within their respective classes. This encapsulation ensures that these elements interact in a controlled and organized manner, simplifying modifications and extensions.
- **Inheritance**: We will create subclasses for different types of obstacles (e.g., buildings, vehicles, barriers, animals, and stones), inheriting properties and behaviors from a common parent class. This approach minimizes code duplication and improves the game's modularity and extensibility.

- **Polymorphism:** Polymorphism will be used to allow different obstacles to exhibit distinct behaviors and interactions with the player's vehicle. For example, vehicles may behave differently from barriers when collided with.
- **Abstraction:** Various design patterns, like the observer pattern for tracking health changes or the factory pattern for obstacle generation, will be employed. These patterns abstract complex processes and enable efficient management of game components.

External Libraries:

To realize the "*Pakistan Drift* " project, we will leverage the Simple Direct Media Layer (SDL) 2.0, a powerful cross-platform development library for C++. SDL provides low-level access to essential hardware components, such as graphics, audio, keyboard, mouse, and joystick.

In addition to the core SDL library, we will utilize the following SDL add-on libraries:

- **SDL Image:** This library will enable us to load and manipulate various image formats, including PNG, JPEG, BMP, and GIF. It provides functions to create, manipulate, and convert image surfaces, enriching the visual elements of the game.
- **SDL Mixer:** With SDL Mixer, we can incorporate a wide range of audio formats (WAV, MP3, OGG, MIDI) into the game. It allows us to load and play audio files, control sound parameters, and create immersive audio experiences.
- **SDL ttf:** SDL ttf will provide support for rendering TrueType fonts, allowing us to display text on the screen with control over font style, size, and color. This enhances the visual elements and user interface of the game.

Synopsis of the UML diagram:

Class	Attributes	Methods	Description
Unit	src_rect	Move	Parent class for all obstacles, providing a common interface.
FactoryClass	objects: List<Unit*>	createObjects(Unit), ~FactoryClass()	Manages the creation of game objects.
CarLogic	playercar: PlayerCar, factory: FactoryClass, scores: int, objects: List<Unit*>, SCREEN_WIDTH: const int, SCREEN_HEIGHT: const int	Checkup(), Checkdown(), Checkleft(), Checkright(), objectcheck(), passed_obstacle(), drawObjects(), createObjects(int, int), checkCollision(), updateScore(), ~CarLogic()	Manages game logic related to player cars and obstacles.
PlayerCar	srcRect, moverRect	Draw(), PlayerCar(), ~PlayerCar(), operator++(),	Represents the player's car and handles various car-related operations.

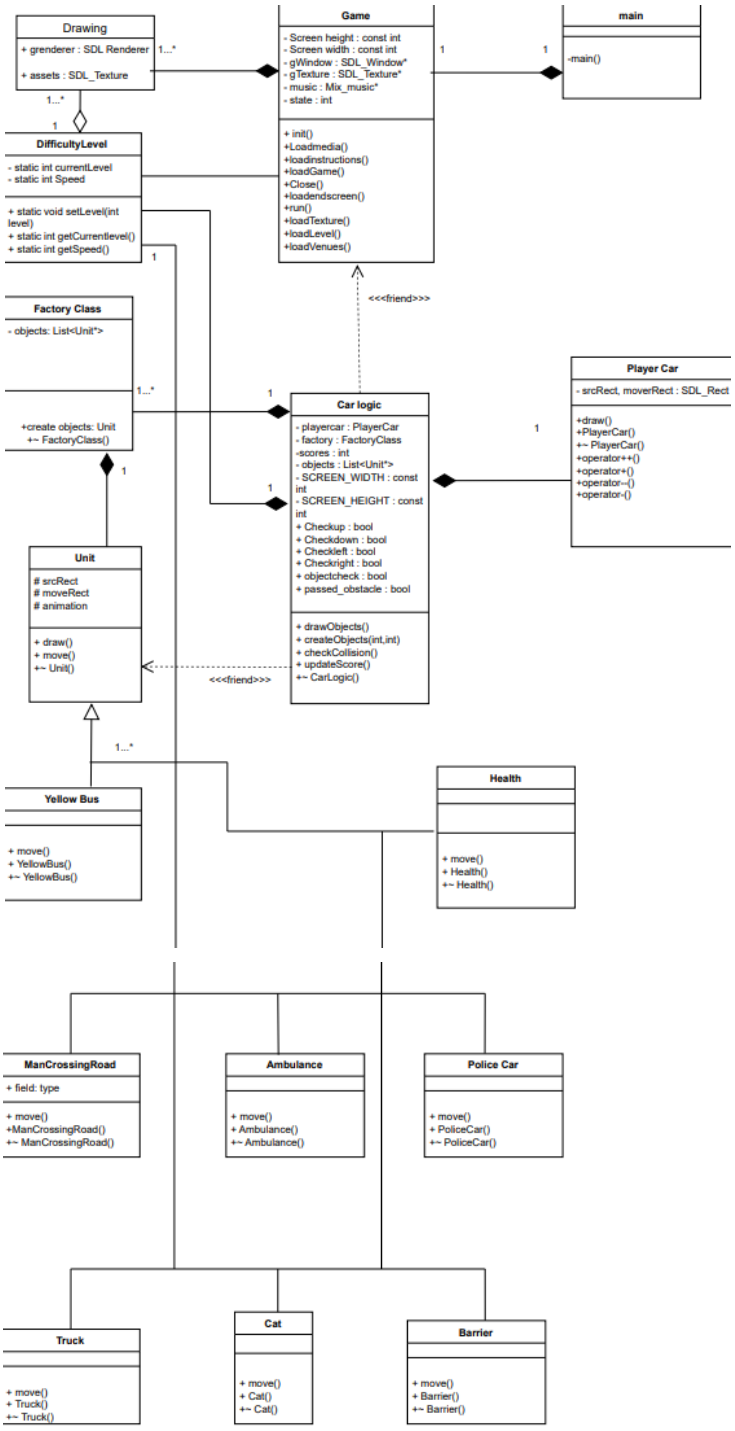
		operator+(), operator--(), operator-()	
Drawing	renderer: SDL_Renderer, assets: SDL_Texture		Manages drawing elements in the game.
Main		main()	Entry point for the game.
Game	Screen height: const int, Screen width: const int, gWindow: SDL_Window*, gTexture: SDL_Texture*, music: Mix_music*, state: int	init(), Loadmedia(), loadinstructions(), loadGame(), Close(), loadendscreen(), run(), loadTexture(), loadLevel(), loadVenues()	Manages the game's overall state, media, and provides game functionality.
Background	Renderer	Draw	Handles dynamic background rendering.
HighScore	current_score, list of all scores	Loading_score, Saving_score, Adding_score, Displaying_high_score, Deleting_score	Tracks and displays player scores.
Displaying_Game_Information	renderer, score_texture, lives_texture, speed_texture	Displaying_score, Displaying_lives, Displaying_speed, Managing_speed, Managing_lives, Managing_speed	Displays game information, such as scores, lives, and speed.
Keyboard_Input	flag_backward_move, flag_forward_move, flag_upward_move, flag_downward_move	Input_manager	Manages player input from keyboards.
Game_Over	m_renderer, backgroundTexture, backgroundRect, textRect, player score	Gameover()	Manages the Game Over screen display.
Health		move(), Health(), ~Health()	Represents the player's health status.
Car_obstacles	initial_x_value, initial_y_value	Moving	Represents car obstacles on the road.
Wood_obstacles	initial_x_value, initial_y_value	Moving	Represents wooden obstacles on the road.
Rock_obstacles	initial_x_value, initial_y_value	Moving	Represents rock obstacles on the road.
Truck	srcRect, moverRect	Move()	Represents a truck

			obstacle on the road.
Cat	srcRect, moverRect	Move()	Represents a cat obstacle on the road.
Barrier	srcRect, moverRect	Move()	Represents a barrier obstacle on the road.
ManCrossingRoad	srcRect, moverRect	Move()	Represents a man crossing the road.
Ambulance	srcRect, moverRect	Move()	Represents an ambulance obstacle on the road.
PoliceCar	srcRect, moverRect	Move()	Represents a police car obstacle on the road.
YellowBus	srcRect, moverRect	Move()	Represents a yellow bus obstacle on the road.

Challenges Faced:

- **Number of Classes:** Our first challenge was in determining the number and types of classes to be used in project
- **Sprite Pack and Game Screen:** Now since our project is game we needed game screens that improve the overall user interface (UI), so developing game screens and a diverse sprite pack that encompasses all in-game elements, such as vehicles and obstacles was tough.

UML:


Drawing
+ grenderer : SDL Renderer
+ assets : SDL_Texture
1...*
1
Game
- Screen height : const int
- Screen width : const int
- gWindow : SDL_Window*
- gTexture : SDL_Texture*
- music : Mix_music*
- state : int
+ init()
+Loadmedia()
+loadinstructions()
+loadGame()
+Close()
+loadendscreen()
+run()
+loadTexture()
+loadLevel()
+loadVenues()
main
-main()
DifficultyLevel
- static int currentLevel
- static int Speed
+ static void setLevel(int level)
+ static int getCurrentlevel()
+ static int getSpeed()
<<<friend>>>
Factory Class
- objects: List<Unit*>
+create objects: Unit
+~ FactoryClass()
Car logic
- playercar : PlayerCar
- factory : FactoryClass
-scores : int
- objects : List<Unit*>
- SCREEN_WIDTH : const int
- SCREEN_HEIGHT : const int
+ Checkup : bool
+ Checkdown : bool
+ Checkleft : bool
+ Checkright : bool
+ objectcheck : bool
+ passed_obstacle : bool
+ drawObjects()
+ createObjects(int,int)
+ checkCollision()
+ updateScore()
+~ CarLogic()
Player Car
- srcRect, moverRect : SDL_Rect
+draw()
+PlayerCar()
+~ PlayerCar()
+operator++()
+operator+()
+operator--()
+operator-()
Unit
srcRect
moveRect
animation
+ draw()
+ move()
+~ Unit()
Yellow Bus
+ move()
+ YellowBus()
+~ YellowBus()
Health
+ move()
+ Health()
+~ Health()
ManCrossingRoad
+ field: type
+ move()
+ManCrossingRoad()
+~ ManCrossingRoad()
Ambulance
+ move()
+ Ambulance()
+~ Ambulance()
Police Car
+ move()
+ PoliceCar()
+~ PoliceCar()
Truck
+ move()
+ Truck()
+~ Truck()
Cat
+ move()
+ Cat()
+~ Cat()
Barrier
+ move()
+ Barrier()
+~ Barrier()