Object-Oriented Programming and Design Methodologies

UML and PROJECT SYNOPSIS

“Geometry Dash”

CS/CE 224/272, T6, FALL 2023, HABIB UNIVERSITY

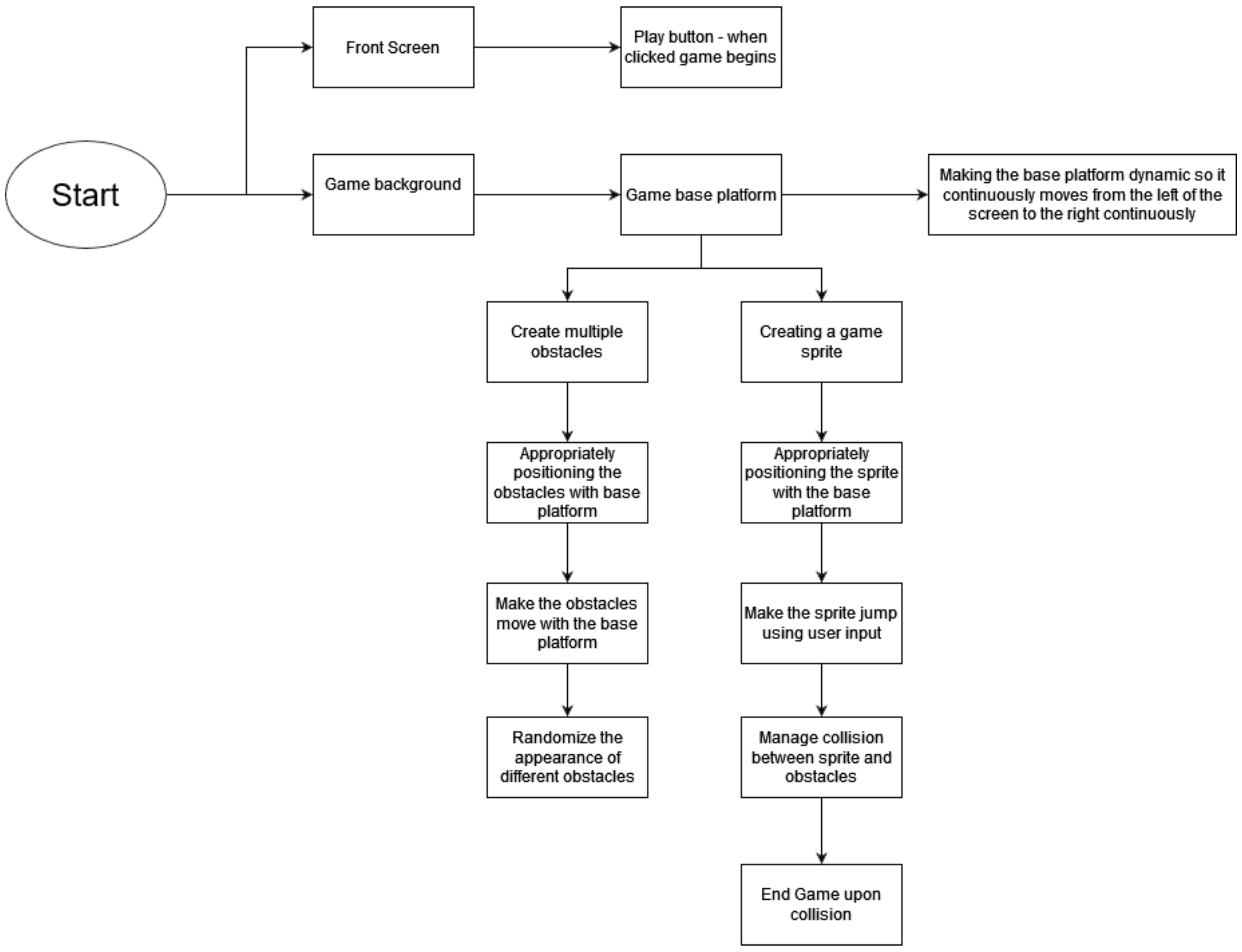
Instructor | Dr. Musabbir Majeed

Abdullah Amir, 08453 | Ehzem Sheikh, 08518 | Muhammed Aqeel, 08442

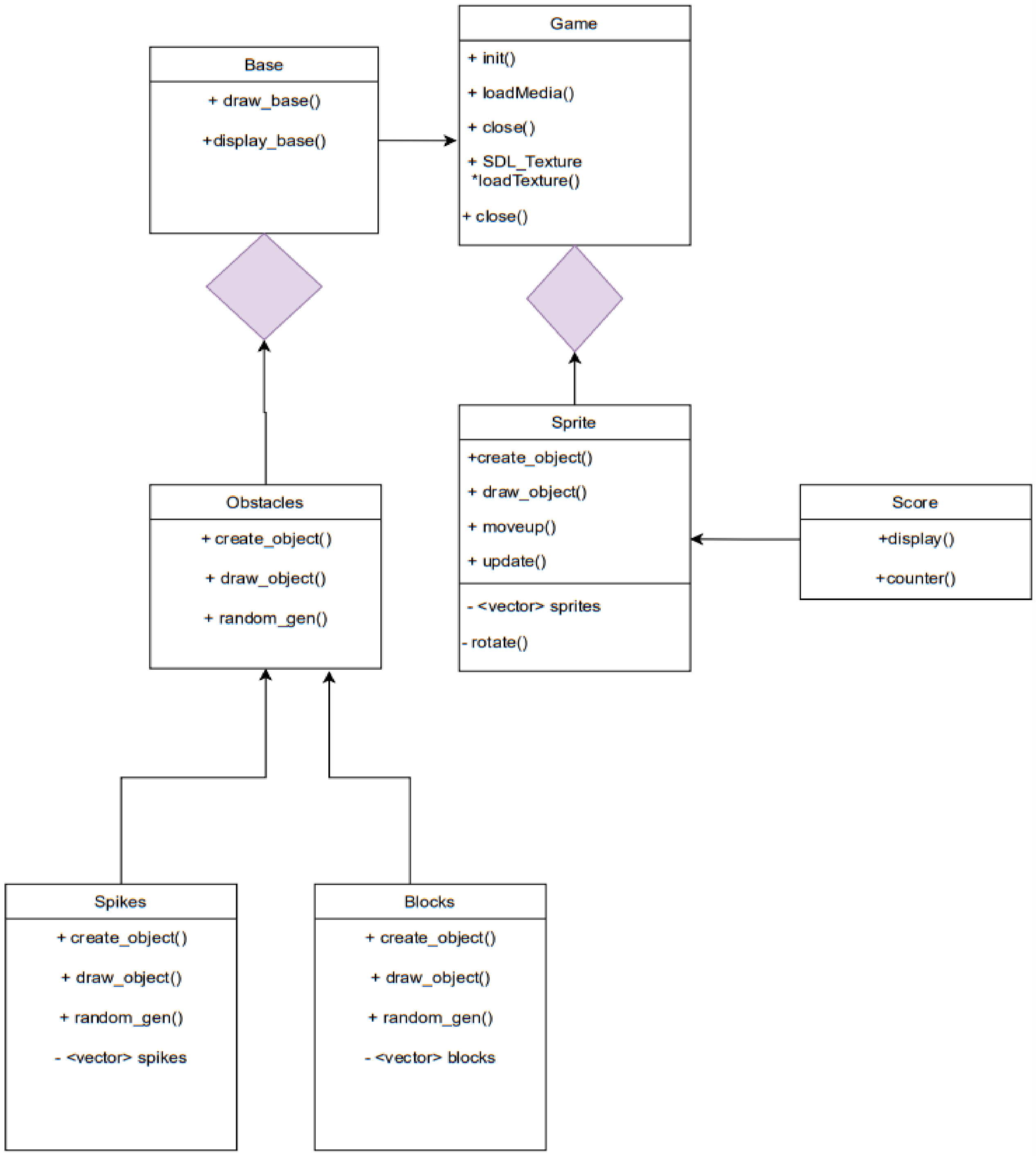
Project Overview:

Our aim is to recreate a classic, thrilling, fast-paced 2D platformer game, inspired by the popular mobile game “Geometry Dash”. The gameplay is based around moving a sprite over a series of obstacles, spikes, and other creative hazards across increasingly challenging levels. The objective will be to ensure the object reaches the end without colliding with any of the hazards.

Project Overview:


Start
Front Screen
Play button - when clicked game begins
Game background
Game base platform
Making the base platform dynamic so it continuously moves from the left of the screen to the right continuously
Create multiple obstacles
Creating a game sprite
Appropriately positioning the obstacles with base platform
Appropriately positioning the sprite with the base platform
Make the obstacles move with the base platform
Make the sprite jump using user input
Randomize the appearance of different obstacles
Manage collision between sprite and obstacles
End Game upon collision

UML Diagram:


Game
+ init()
+ loadMedia()
+ close()
+ SDL_Texture *loadTexture()
+ close()
Base
+ draw_base()
+display_base()
Sprite
+create_object()
+ draw_object()
+ moveup()
+ update()
- <vector> sprites
- rotate()
Score
+display()
+counter()
Obstacles
+ create_object()
+ draw_object()
+ random_gen()
Spikes
+ create_object()
+ draw_object()
+ random_gen()
- <vector> spikes
Blocks
+ create_object()
+ draw_object()
+ random_gen()
- <vector> blocks

Description:

Base Class:
The Base class serves as a fundamental component. It encapsulates basic functionality for drawing and displaying a base element. The drawBase() function is responsible for rendering the base, and displayBase() handles the presentation of the base element.

Game Class:
The Game class plays a pivotal role in managing the overall game state. It is associated with the Base class, indicating a relationship where the Game class utilizes functionalities from the Base class.

Functionality within the Game class includes:
init(): Initializes the game, setting up necessary components.
loadMedia(): Loads game media, such as textures or images.
close(): Handles the closure of the game, freeing up resources.
loadTexture(): Loads an SDL texture, potentially used for rendering.

Sprite Class:
The Sprite class is intricately tied to the Game class through a composition relationship, suggesting that a Sprite is a vital part of the Game.

Functionalities encapsulated by the Sprite class include:
createObject(): Creates a sprite object.
drawObject(): Renders the sprite object on the screen.
moveUp(): Initiates an upward movement of the sprite.
update(): Manages the update logic of the sprite.
rotate(): Enables rotation of the sprite.

Obstacle Class:
The Obstacle class inherits from the Base class, indicating that an obstacle is a specialized form of the base element. This relationship implies that the Obstacle class inherits basic functionalities from the Base class.

Core functionalities of the Obstacle class encompass:
createObject(): Generates an obstacle object.
drawObject(): Renders the obstacle object on the screen.
randomGenerator(): Generates random obstacles.
Additionally, the class has a composition relationship with the Base class, implying that an obstacle is composed of or contains a base element.

Spikes Class:
The Spikes class inherits from the Obstacle class, signifying that spikes are a specialized form of obstacles. This relationship implies that the Spikes class inherits functionalities from the Obstacle class.
The class maintains a vector (spikes) to manage multiple spike instances.

Blocks Class:
The Blocks class inherits from the Obstacle class, signifying that blocks are a specialized form of obstacles. This relationship implies that the Blocks class inherits functionalities from the

Obstacle class:
The class maintains a vector (blocks) to manage multiple block instances.
This updated description reflects the additional details about the inheritance relationships between the Obstacle, Spikes, and Blocks classes, as well as the composition relationship between the Obstacle class and the Base class.

Graphics and Screens:

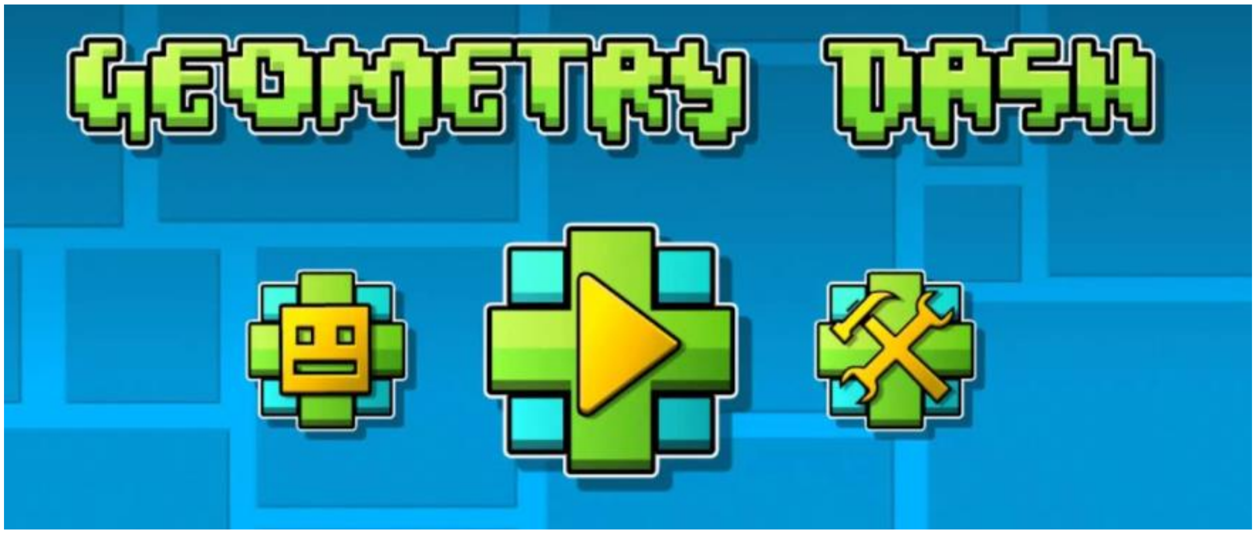


Figure 1

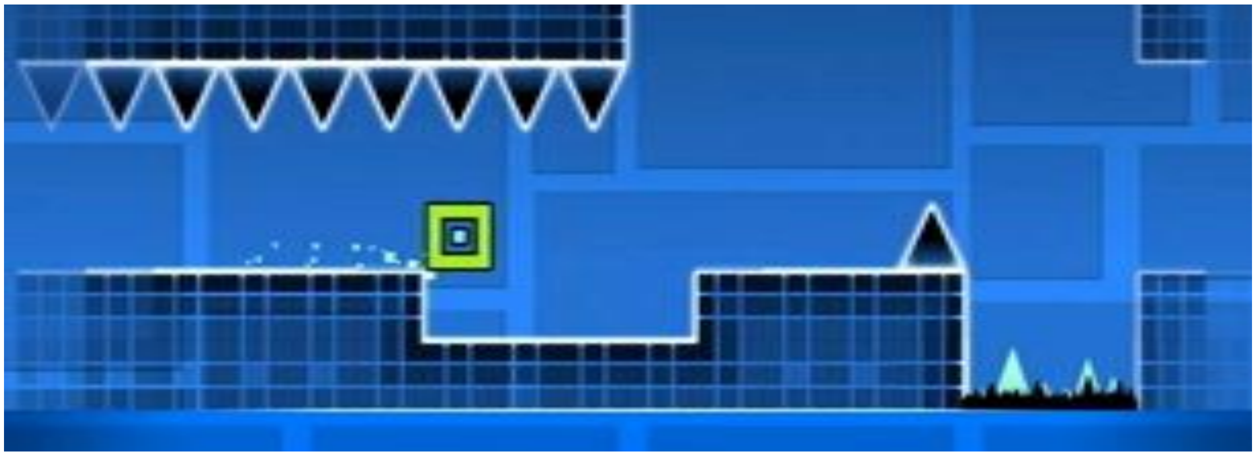


Figure 2

Figure 1 shows the start screen of our game. It shows the name of the game and a play button in the center of the screen. Upon clicking the play button game play is initiated.

Figure 2 illustrates a carefully designed composition which includes a variety of obstacles such as spikes and moving platforms, a sprite and platform base. The game sprite, with responsive controls, engages in precise maneuvers. The visual dynamism, enhanced by dynamic backgrounds and animations, contributes to the immersive experience. Though not shown explicitly, a scoring mechanism likely encourages replayability.