UTM

UNIVERSITI TEKNOLOGI MALAYSIA

FACULTY OF COMPUTING
PROGRAMMING TECHNIQUE II
(SECJ1023)

FINALE PROJECT

SEMESTER II 2024/25

STUDENT 1 : **CHUN YAO TING (A24CS0239)**

STUDENT 2 : **HENG ZHI QIANG (A24CS0081)**

STUDENT 3 : **GAVENESH A/L BATUMALAI (A24CS0076)**

STUDENT 4 : **RYAN SEGAL ANAK NICKLAS (A22BS0118)**

SECTION / GROUP : **SECTION 02 / TYPESTRIKE**

LECTURER NAME : **DR. SHAMINI A/P RAJA KUMARAN**

Table of Contents

1.1 Project Deliverable 1: Project Proposal

1.1.1 Project Idea

The purpose of this project is to create a fast type fighting game where a player competes against a computer opponent in an attempt to emerge victorious. The game features an innovative system where each word that a player types can be used as a "bullet" to hit the computer foe. The end result is to deplete the opponent's health bar to zero before the player's own health bar is depleted.

The game generates a random string of words, or "strings," that must be typed correctly and with speed to give the player ammunition. For each accurately input string, damage is inflicted on the computerized enemy, and faulty or slow input exposes the player to counterattack. The game is based on a duel where the player's typing speed and accuracy have a direct influence on their success in combat.

The health bar feature adds an element of tension and strategy, emphasizing the requirement for both offense (speed and accuracy of typing) and defence (evading errors and speed reductions). The instant a player's health bar is completely depleted, that player loses.

The mission of this project is to merge entertainment and skill-building, in this case typing speed and accuracy, with random string generation, event-driven programming, and real-time UI updating. A few potential future improvements include difficulty levels, visual effects, sound effects, and leaderboards.

1.1.2 Key Findings from Existing System

- **Typing of the Dead (by Sega)**

  Description: A spin-off of the arcade shooter "House of the Dead," this game replaces guns with typing challenges to eliminate zombies.

  Similarity: Words appear as enemies and typing them defeats the foes. Features a health bar and progressive difficulty.

- **Epistory – Typing Chronicles**

  Description: A story-driven action-adventure game where players type to cast spells, solve puzzles, and fight enemies.

  Similarity: Combines typing mechanics with combat and exploration, featuring health and progression systems.

- **Nitro Type**

  Description: A competitive typing game where players race against others by typing words accurately.

  Similarity: Typing speed and accuracy determine victory; indirect combat via racing.

- **God of Word**

  Description: A typing-based gladiator combat game set in ancient mythology. Players face off in typing duels with various enemies.

  Similarity: Real-time combat, health bars, random word challenges, and power-ups.

1.1.3 Game Mission

I. Practice Typing Speed in a Fun Way

The game transforms traditional typing drills into an engaging experience by integrating gameplay mechanics. Players improve their typing speed and accuracy through typing battles, where correctly typed words act as "bullets" to attack opponents. Elements like dynamic word generation, health bars, and power-ups (e.g. DOUBLE DAMAGE, HEALING) create a rewarding loop: faster and more accurate typing directly translates to in-game success. This gamified approach keeps users motivated, blending skill development with excitement.

II. Kill the Opponent

The competitive objective of defeating the opponent adds urgency and stakes to typing practice. Players must outpace their rival by typing words quickly and accurately to deplete the enemy's health bar. This head-to-head combat mechanic ensures that typing practice is not isolated but contextualized within a thrilling duel, making skill improvement a natural outcome of gameplay.

1.1.4 Game Style

I. Competitive (Between 2 Players)

The game emphasizes real-time competition, where two players face off in a typing duel. This setup fosters a sense of rivalry, encouraging players to refine their skills to dominate matches. Features like live scoreboards, health-bar comparisons, and spells (e.g., MANIPULATION to sabotage opponents) amplify the competitive spirit, ensuring each match is dynamic and replay ability.

II. 2D Game Interface

The 2D design prioritizes simplicity and clarity, allowing players to focus on typing without visual clutter. The interface includes retro-styled "laptop" avatars, health bars, and attack animations, in a flat and accessible style. This minimalist approach ensures smooth gameplay and intuitive navigation, ideal for both casual and serious gamers.

1.1.5 Benefits of TypeStrike

There are several benefits brought by TypeStrike, which makes it outstanding from existing competitors. First, TypeStrike improves typing skills in an engaging way. Players naturally enhance their typing speed and precision as they race to attack opponents with "word bullets." The faster and more accurately they type, the more damage they deal. It also provides real-time feedback where immediate in-game consequences (e.g., health bar depletion, spell effects) provide tangible feedback, motivating users to refine their skills.

Secondly, we introduced gamified practice which makes learning fun and competitive. TypeStrike replaces monotonous typing drills with thrilling duels, where typing becomes a weapon. Users stay engaged through dynamic battles rather than passive exercises. Besides, competing against friends or other players adds excitement and encourages playability. The desire to "win" drives consistent practice.

Thirdly, it also encourages healthy habits. Regular play sharpens cognitive skills like focus, hand-eye coordination, and quick decision-making. Besides, the fast-paced, competitive nature serves as a fun outlet for stress, blending productivity with entertainment.

1.1.6 Storyboard of TypeStrike

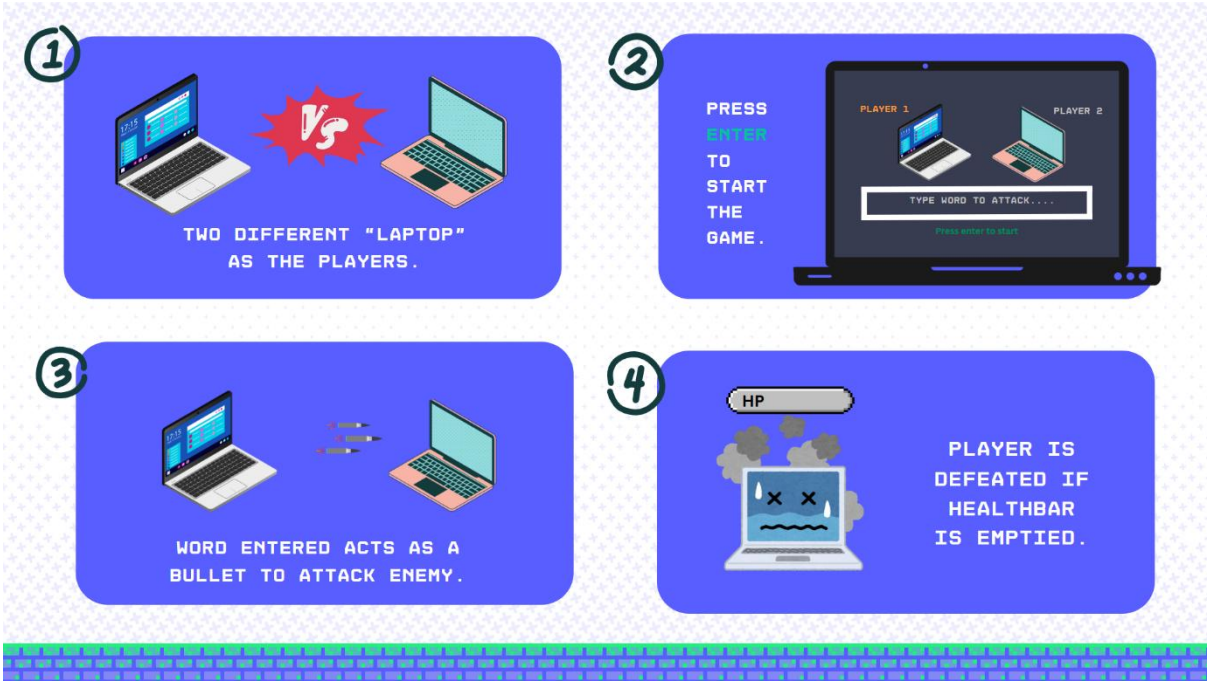

Figure 1: Storyboard of Game

Table 1: Flow of The Game

No	Flow	Description
1.	**Two Players, Two "Laptops"**	Two retro-styled laptop avatars represent Player 1 and Player 2. This visualizes the competitive duel, setting the stage for a head-to-head typing battle.
2.	**Words as Bullets**	Players attack by typing words displayed on-screen. Each correctly typed word acts as a "bullet" to damage the opponent's health bar. This mechanic ties typing accuracy and speed directly to gameplay success
3.	**Game Start & Controls**	The match begins when players press ENTER. A prompt instructs players to "TYPE WORD TO ATTACK...", emphasizing the real-time action. Players must type rapidly to outpace their rival.
4.	**Health Bar & Defeat**	Both players have HP (health bars) displayed prominently. A player is defeated when their health bar is fully depleted, ending the match. This creates urgency and stakes, rewarding faster, more accurate typists.

1.1.7 Game Interface

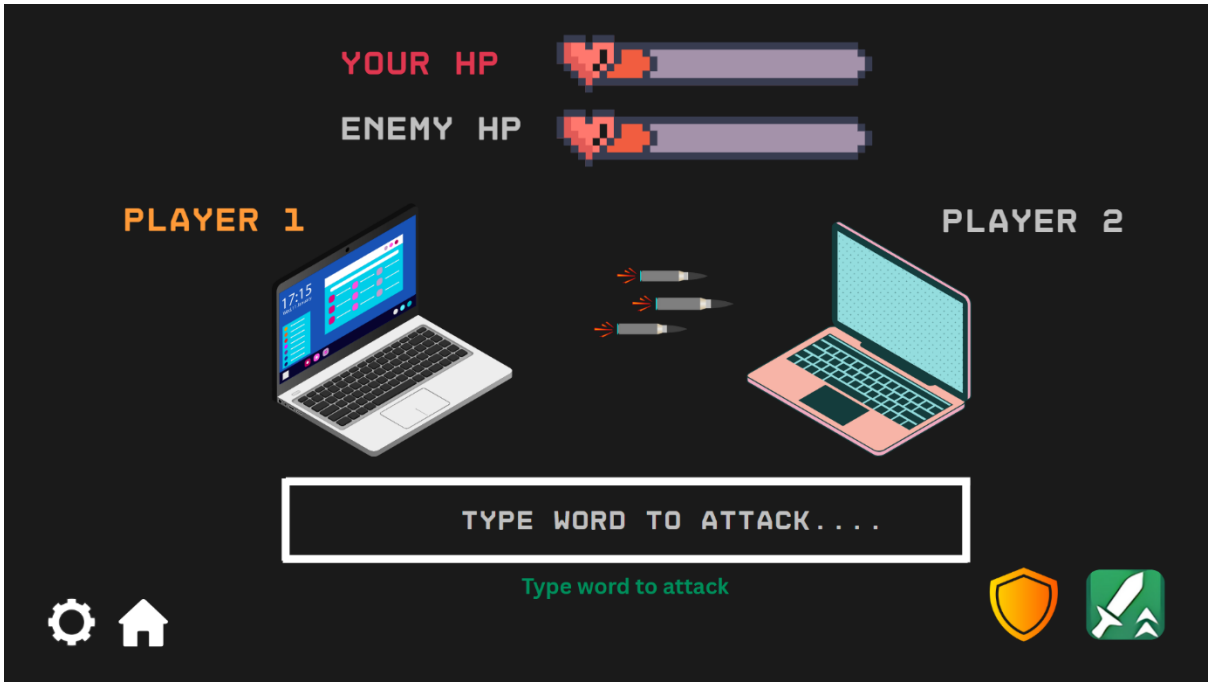

Figure 2: Fighting Interface

Figure 3: Game Over Interface

1.1.8 Extra Features

To make our game more fun, we have proposed more extra features to improve the playability of the game. We introduced the idea of Spell, which is inspired by mobile games like Mobile Legends or Honor of Kings, where the players need to choose two spells before the game starts, while each spell have their distinct functionalities. However, the different part is our game doesn't support reuse of spell, meaning that the game's spell is in one-time casting mode, because we don't want our game to be extensively long due to the unlimited reuse of spells. Other than that, instead of knowing the opponents' spell before the game starts like existing games, we make it not visible to opponents. This mean that you won't know what spells enemy brought, and enemy also doesn't know what spells you brought, leading to a more exciting mind game between players.

In TypeStrike, currently we proposed four types of spells, all with unique functions. First, we have double damage which doubles your damage dealt towards enemy for 10 seconds. This will function as an offensive spell to shrink down enemy's health quickly. Secondly, the healing spells provide healing effect for 15 seconds based on how many words you typed, while reducing the damage dealt by the enemy for 50%. We added the reduce damage features as your health will not be healing if the enemy just keep attacking you and cancel out your healing effects. This spell will acts as a defensive spell to come back from the dying state. Thirdly, the Invincible spell will dismiss all damage dealt by enemy for 8 seconds. This is considered a very defensive spell to prevent your health from shrinking for a short period. Fourthly, the Booster spell is an offensive spell, where player deals 200% of damage towards enemy if they typed the words correctly for certain streak. This spell can only be activated when the player typed the words correctly for 5 times, and the spell is immediately deactivated once the correct streak is gone.

1.2 Project Deliverable 2: Project Analysis & Design

1.2.1 Object & Classes

Table 2: Classes with Their Members

	Class	Members	
		Attributes	**Methods**
1	Game	-player1: Player -player2: Player -dictionary: vector<string> -wordPool: WordPool -gameRunning: bool -currentInput: string -renderer: GraphicsRenderer* -spellNotificationTime: time_point -showSpellNotification: bool -currentSpellNotification: string -lastBotAttack: time_point -botDamageInterval: int -botDamageAmount: int -playerDamagePerWord: int -lastFrameTime: time_point -TARGET_FPS: const int = 30	+Game(p1Name: const string& , existingRenderer: GraphicsRenderer*) +~Game(): +loadDictionary(): void +updateBotAI(): bool +processWord(word: const string&): void +gameLoop(): void +getWPM(): double +getPlayer(): Player&

		-FRAME_TIME_MS: const int = 1000 / TARGET_FPS	
2	GraphicsRenderer	-screenWidth: int -screenHeight: int -lastInput: string	+GraphicsRenderer(width: int= 800, height: int= 600) +~GraphicsRenderer() +initializeStars(): void +drawHealthBar(x: int, y: int, width: int, height: int, percentage: double, color: int, label: const string&): void +drawPixelLaptop(x: int, y: int, color: int): void +drawRobot(x: int, y: int): void +drawBullets(startX: int, startY: int, endX: int, endY: int): void +updateInputArea(currentInput: const string&): void +drawSpellNotification(spellName: const string&): void +drawGameInterface(playerHP: double, enemyHP: double, words: const vector<string>&, currentInput: const string&, playerScore: int, accuracy: double, forceRedraw: bool= false): void +void drawGameOver(playerWon: bool, finalScore: int, accuracy: double, wpm: double, cpm: double, totalTimeSeconds: double): void +isKeyPressed(): bool +getKey(): char
3	WordPool	-allWords: vector<string> -currentIndex: size_t	+WordPool() +loadWords(dictionary: const vector<string>&): void +getCurrentWords(count: int= 10): vector<string>

			+checkCurrentWord(word: const string&): bool +isEmpty(): bool +getRemainingWords(): int +getCurrentWord(): string
4	Player	-name: string -healthbar: HealthBar -score: int -correctWords: int -totalWords: int -totalCharactersTyped: int -spells[2]: Spell* -gameStartTime: time_point -gameEndTime: time_point -gameEnded: bool	+Player(playerName: const string&, maxHp: int= 100): +takeDamage(int damage): void+heal(int amount): void +addScore(int points): void +incrementCorrectWords(): void +incrementTotalWords(): void +addCharactersTyped(chars: int): void +endGame(): void +isAlive(): bool +getName(): string +getScore(): int +getHealthBar(): HealthBar& +getAccuracy(): double +getTotalTimeSeconds(): double +getWPM(): double +getCPM(): double +~Player() +setSpell(index: int, spell: Spell*): void +activateSpell(index: int): void +updateSpells(): void +getSpell(index: int): Spell*

5	Spell	#name: string #desc: string #duration: int #isActive: bool #remainingTime: int #isUsed: bool #spellType: SpellType	+Spell(n: string, d: string, dur: int, type: SpellType) +~Spell(): virtual +activate(player: Player&): virtual void = 0 +update(player: Player&): virtual void = 0 +deactivate(): virtual void +getDamageMultiplier(): virtual double +getDamageReduction(): virtual double +isImmune(): virtual bool +applyWordHeal(player: Player&): virtual void +addCorrectWord(): virtual void +resetStreak(): virtual void +getCurrentStreak(): virtual int +decrementTime(): void +checkStatus(): bool +getRemainingTime(): int +canUse(): bool +markAsUsed(): void +getName(): string +getType(): SpellType
6	HealthBar	-maxHealth: int -currentHealth: int	+HealthBar(maxHp: int= 100) +takeDamage(damage: int):void +heal(amount: int): void +isAlive(): bool +getCurrentHealth(): int +getMaxHealth(): int +getHealthPercentage(): double

7	Healing<Spell>	-healPerWord: int -dmgReduction: double	+Healing() +activate(player: Player&): void +update(player:Player&): void +getDamageReduction() : double +applyWordHeal(player: Player): void
8	Booster<Spell>	-dmgMulti: double -requiredStreak: int -currentStreak: int -hasFailed: double	+Booster() +update(player:Player&):void +activate(player: Player&): void +addCorrection(): void +resetStreak(): void +getDamageMultiplier() : double +getCurrentStreak(): int
9	DoubleDamage<Spell>	-dmgMulti: double	+DoubleDamage(): +activate(player:Player):void +update(player: Player): void +getDamageMultiplier() const:double
10	Invincible<Spell>		+Invincible(): +activate(player:Player):void +update(player: Player): void +isImmune() const: bool

1.2.2 Relationship Between Classes

It is important to understand the relationship between classes before we start to develop the actual program. First of all, the Game Class is composed of three major components. The first one is Player, and the second one is WordPool, and the third one is GraphicsRenderer. The Player Class is having a strong association relationship with Game Class, as the Game straight away ends when Player is destructed. Therefore, we can see it is having a 1 to 2 composition relationship between Game and Player, as a game contains two players, which are the player and the opponents. Other than Players, the game also uses WordPool to generate random words for the input by user. Without WordPool, there is no way the game can run like normal. Therefore, we can conclude that WordPool have a composition relationship with Game. Last but not least, the GraphicsRenderer have aggregation with Game to draw the game interface.

Next, we can observe few important of components for Player Class. First, each of the Player Class must have Healthbar Class to track their current health. This is a composition because the HealthBar must be there for each player to stay alive. Other than that, a Player can activate Spell during the gameplay. Each Player can have zero to maximum two active Spell at any time. This is an aggregation because Spell class affect Players but are separate game entities with their own behaviours and lifespan, and Spell and Player at exists on their own, as Player does not necessarily have a Spell. We also use an array of object of Spell for two spells.

Meanwhile, the class such as Double Damage, Booster, Healing, and Invincible is having the relationship of inheritance and polymorphism with Spell. Spell Class is designed to be as generalized as possible, while the respective spells inherited the basic functions of Spell and designed to be more specified to handle certain functions. Additionally, the Healing Class is having dependency with Word Class, as the healing amount depends on how many words you've typed correctly. Other than Healing Class, the Booster Class is also having dependency with Word Class, because Booster spell required certain number of words to activate the streak. With just a single wrong word after you activated Booster, the streak will be immediately deactivated. Based on its characteristics, we can see it is highly dependent on the WordPool.

1.2.3 UML Class Diagram

1.2.3.1 Overview of UML Class Diagram

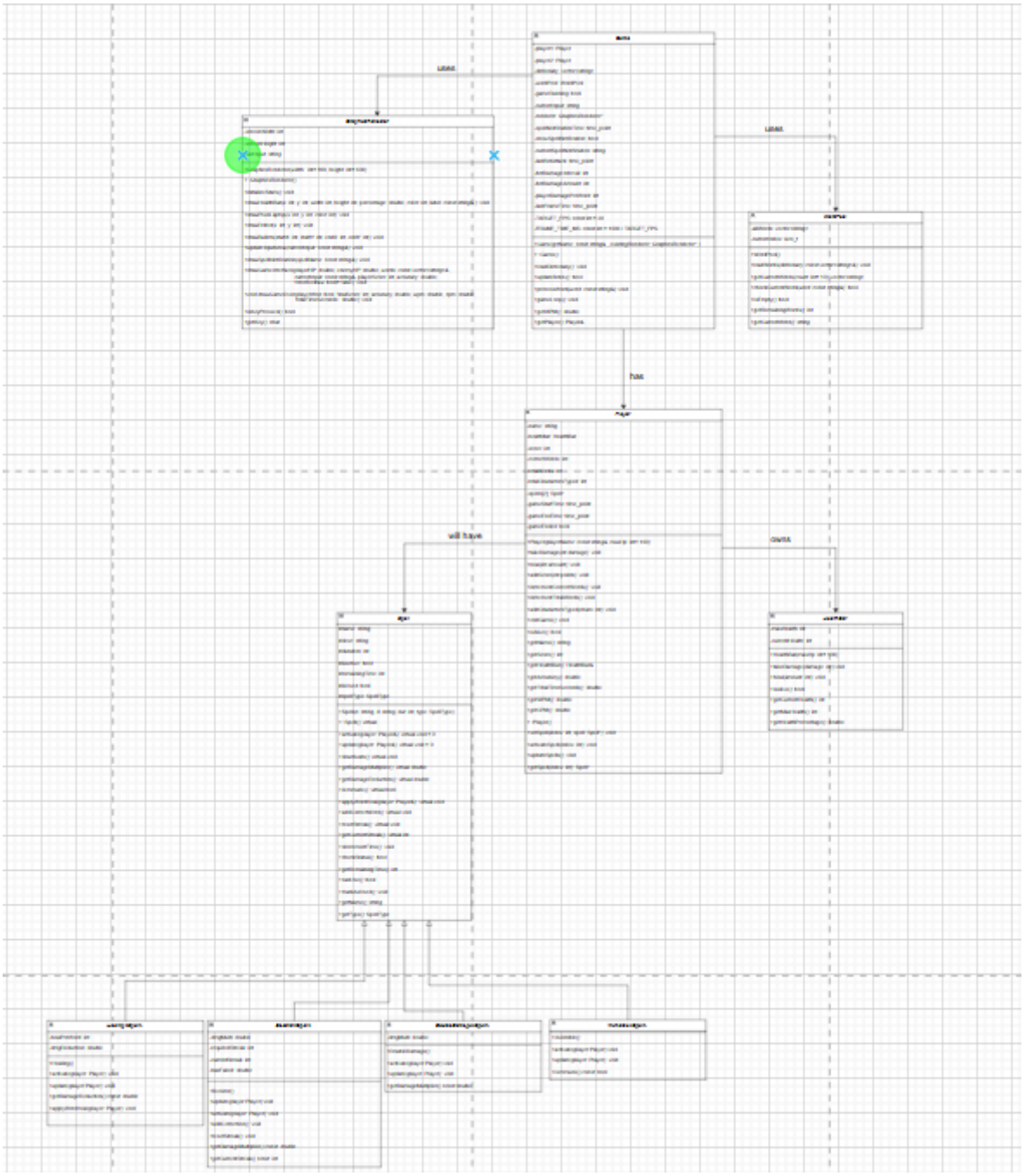

Figure 4: Overview of TypeStrike UML Diagram

1.2.3.2 Game Association

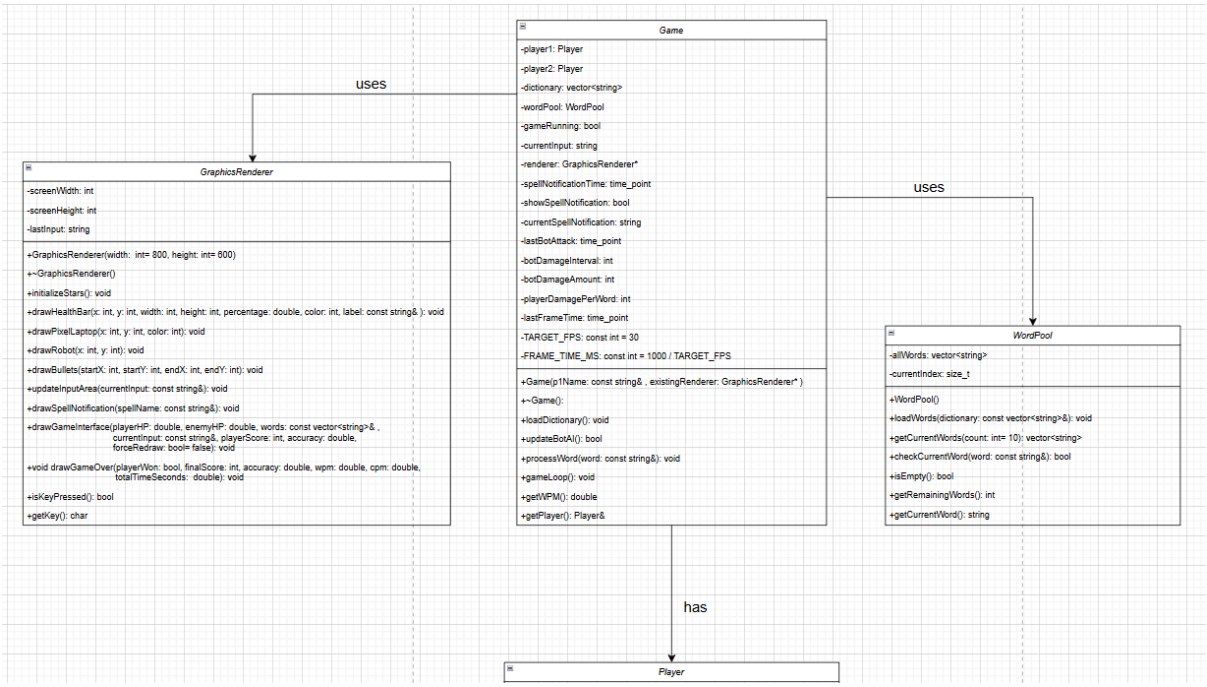

Figure 5: Game Association UML Diagram

1.2.3.3 Player Association

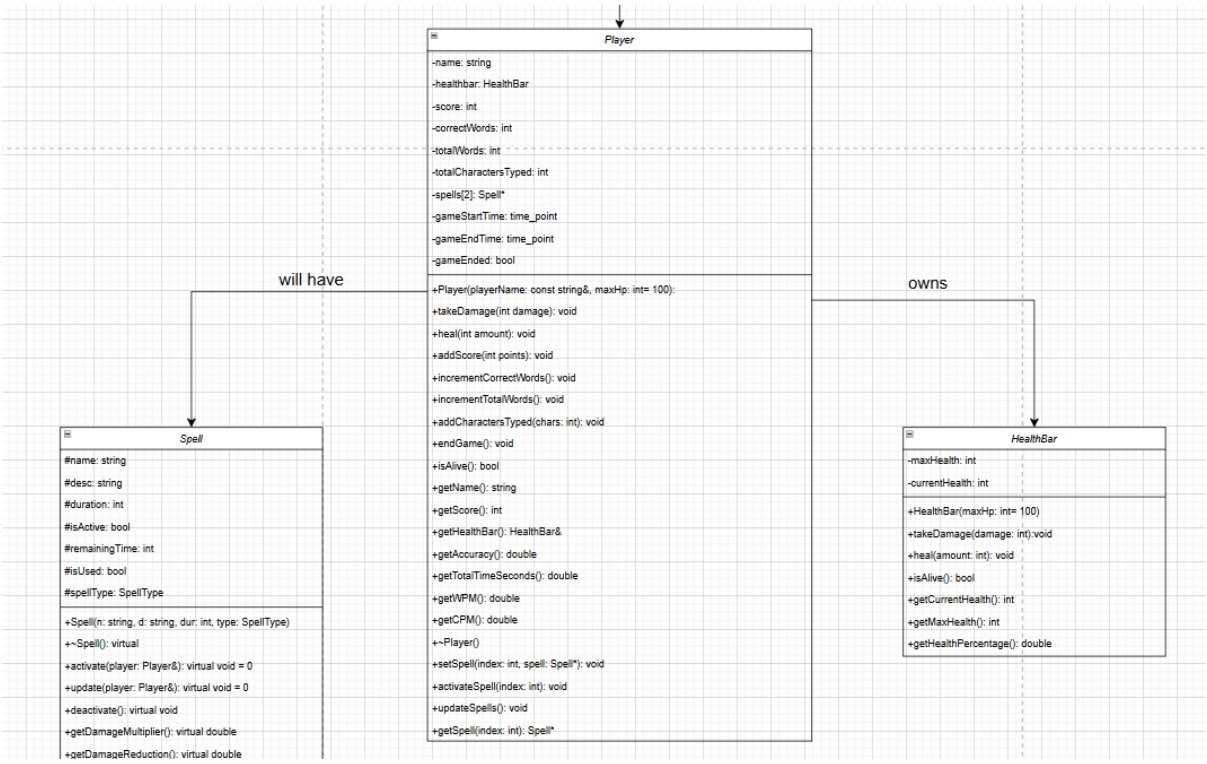

Figure 6: Player Association UML Diagram

1.2.3.4 Spell Inheritance

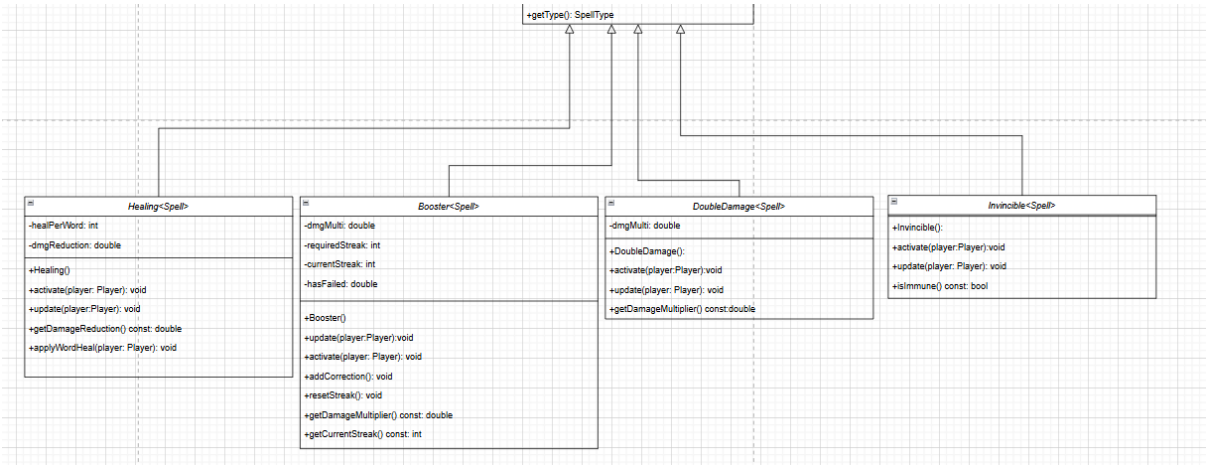

Figure 7: Spell Inheritance Relationship UML Diagram

APPENDICES

1) First Presentation Slide Link

https://www.canva.com/design/DAGkx1j5ugE/CNtLKhSp42ucxoxLdc0J-A/edit?utm_content=DAGkx1j5ugE&utm_campaign=designshare&utm_medium=link2&utm_source=sharebutton

2) Brainstorming Video of Project Deliverable 2

https://youtu.be/1Hp9-Lg4JOo?si=varLBC_Xnt73hxpM

3) UML Diagram Link

https://drive.google.com/file/d/1_dh5l91SuOzBN7BTAKhYcIgX59glC5bq/view?usp=sharing

4) Inspiration

https://www.livechat.com/typing-speed-test/#/

5) Final Presentation Video

https://drive.google.com/file/d/1r34eC1gUKs7CrZklI6AvQi7omArCfABV/view?usp=sharing