## **Overview**

**Game Chosen:** *Tower Defense Game***Target Audience:** People who like classic, strategy, non-progression focused games.

### **Justification**

Tower Defense Games are a popular and common genre of video games, where players place and upgrade **towers** to **defend** against incoming **enemies** from **reaching an** **endpoint**. The problem/challenge lies in designing an **engaging** **gameplay** **experience** that **balances** *difficulty*, *strategy*, and *resource management*.  
  
I chose the tower defense genre because it has **creative freedom** and easy **expandability**, as well as a manageable scope. (simplicity) I have played and enjoyed them in the past (mostly from “Bloons TD6”) and thus have elementary knowledge on how a tower defense game could be designed. To expand upon basic algorithmic logic (sequencing, selection, iteration, recursion etc. – additionally implementing the pygame library), the tower defense game choice further allows me to demonstrate **OOP elements** including:

* **Polymorphism** allowing for the creation of different enemy and tower types, introducing a wider range of attack strategies.
* **Inheritance** allowing for easy implementation of new enemy, tower and map types.
* **Encapsulation** and **Abstraction** by designing of classes that restrict unnecessary data access (private variables) and hide unnecessary internal details.

### **Game Requirements (summary):**

* A Grid Based **Map**/coordinate **system**
* Different **enemy types**, perhaps with unique designs; *health*, *number*, *mechanics*. (e.g. impenetrable to certain tower types)
* **Wave-based** **enemy** **spawning** system, progressively increasing in difficulty.
* **Enemy** spawning and **pathfinding** mechanisms, allowing navigation for enemies from start to end points.
* Different **tower** **types** each with unique designs; *range*, *damage*, *attack* *patterns*/*types*; introduces *strategy* with choice of *distinct* and *unique* towers in different *scenarios*.
* **Resource management system**, enabling **purchasing** and **upgrading** of **towers**; introduces *strategy* with *resource* *management*.
* **Save System** – saves **progress**; *level completion*, *scores*, *settings.*

### Programming Requirements:

* Using OOP (Object Oriented Programming) ensuring use of inheritance, polymorphism, encapsulation, abstraction to ensure code structure, maintainability, and scalability/extensibility.

#### Grid Based Map System

* The game world is represented using a 2D grid (2D array).
* Cells must store information such as walkable paths, tower placements, and obstacles.

#### Enemy Pathfinding & Movement

* Enemies must navigate from a start point to the goal using pathfinding algorithms

#### Tower Placement & Attacks

* Players must be able to place towers only on valid grid cells.
* Towers should have unique attack behaviors (e.g., single-target, splash damage, slow effect).
* Towers should fire projectiles that follow an enemy trajectory or apply area-of-effect (AoE) damage.

#### Enemy Spawning & Wave System

* Enemies spawn in waves, increasing in difficulty.
* Each enemy type has different stats (health, speed, resistance to damage types).

#### Resource & Upgrade System

* Players earn in-game currency for defeating enemies.
* Towers should be upgradable to enhance their damage, range, or special abilities.

#### Game UI & Controls

* The game should have a main menu, start button, settings, and pause/resume controls.
* Players should receive real-time feedback (e.g., HP bars on enemies, damage numbers, notifications for invalid tower placement).

#### Title Screen & Level Management

* The title screen should allow players to start a new game, change settings, or exit.
* The game should support multiple levels/maps, each with unique enemy waves and layouts.

### Example Class Diagrams:

|  |
| --- |
| **Enemy** |
| - reward: int |
| - health: int |
| - speed: float |
| - position: Tuple[int, int] |
| - path: List[Tuple[int, int]] |
| + move(): void |
| + render(): void |
| + take\_damage(): void |
| + is\_dead(): bool |

|  |
| --- |
| **Tower** |
| - damage: int |
| - range: int |
| - fire\_rate: float |
| - cost: int |
| - position: Tuple[int, int] |
| + attack\_enemy(enemy: Enemy): |
| + upgrade(): void |
| + render(): void |

|  |
| --- |
| **Game** |
| - map: Map |
| - towers: List[Tower] |
| - enemies: list |
| - score: int |
| - money: int |
| - current\_wave: int |
| - lives: int |
| + start(): void |
| + update(): void |
| + draw(): void |
| + handle\_events(): void |
| + spawn\_enemy(): Enemy (object) |
| + game\_over(): bool |
| + place\_tower(): Tower |

**Inherits From**

**Inherits From**

|  |
| --- |
| **Tank\_Enemy** |
| - Defense: int |
| + take\_damage(): void |
| + move(): void |
| + render(): |
| + enrage(): void |

|  |
| --- |
| **Flamethrower\_Tower** |
| - Defense: int |
| + attack\_enemy(): void |
| + upgrade(): void |
| + render(): void |

(overriding methods)

(overriding methods)



|  |
| --- |
| **Map** |
| - music: str |
| - sprite: str |
| - grid: list [list[int]] |
| + get\_tile(): int |
| + set\_tile(): void |
| + reset\_map(): void |
| + render(): void |
| + load\_map(): void |

##### Pseudocode Representation of example Tower Class and Flamethrower\_Tower Subclass

CLASS Tower:

CLASS Flamethrower\_Tower Inherits Tower:

METHOD Initialise(position):

CALL parent constructor Initialise(damage=5, range=3, fire\_rate=0.5, cost=200, position)

SET self.damage = 5

SET self.range = 3

SET self.fire\_rate = 0.5

SET self.cost = 200

SET self.position = position

#Overridden tower stats

ENDMETHOD

METHOD attack\_enemy(enemy):

enemy.take\_damage(self.damage)

#Overridden attack method

ENDMETHOD

METHOD upgrade():

Increase self.damage by 2

Increase self.range by 0.5

Decrease self.fire\_rate by 15% (make it faster)

#overridden upgrade stats

ENDMETHOD

METHOD render():

#Render customized appearance/effects

ENDMETHOD

ENDCLASS

METHOD Initialise(damage, range, fire\_rate, cost, position):

SET self.damage = damage

SET self.range = range

SET self.fire\_rate = fire\_rate

SET self.cost = cost

SET self.position = position

ENDMETHOD

METHOD attack\_enemy(enemy):

enemy.take\_damage(self.damage)

ENDMETHOD

METHOD upgrade():

Increase self.damage by 5

Increase self.range by 1

Decrease self.fire\_rate by 10% (make it faster)

ENDMETHOD

METHOD render():

#Render tower

ENDMETHOD

ENDCLASS