# Design Overview for Descend Below

Name: Ta Quang Tung

Student ID: 104222196

# Table of Contents

[Summary of Program 1](#_Toc141383725)

[Required Roles 3](#_Toc141383726)

[Class Diagram 17](#_Toc141383727)

[Design Patterns 18](#_Toc141383728)

[Singleton 18](#_Toc141383729)

[Strategy 18](#_Toc141383730)

[Template method 19](#_Toc141383731)

[Improvements from D program 20](#_Toc141383732)

[Sequence Diagrams 21](#_Toc141383733)

# Summary of Program

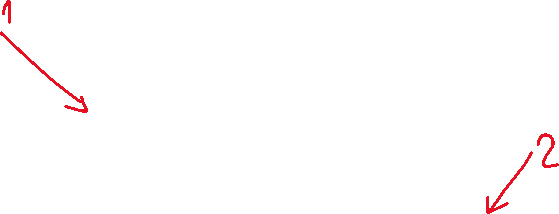
Descend Below is a roguelike video game where the player battles enemies to progress through endless floors. Each floor is made up of rooms, which contain enemies that become stronger the more floors the player progresses. The player gains experience by defeating enemies to level up and become stronger.

The player is equipped with a weapon and attacks by left-clicking. The player can find spells that trigger special effects when cast. Spells, weapons, and healing potions can be found in chests that randomly spawn throughout the game.

On each floor, one room will contain a staircase at the center. Right-clicking the staircase when nearby takes the player to a new floor. The game ends when the player’s health reaches 0, after which it can be reset.

A screenshot of a video game

Description automatically generated



*Figure 1: The final output. Here the player can be seen attacked by enemies (1) while opening a chest whose items are shown on the right (2). The player’s stats are shown on the bottom left (3). The floor information and game instructions are shown on the right (4).*

# Required Roles

The tables below describe the classes/interfaces/enumerations implemented in the project. Items highlighted in red have been added in the HD version. Items that are not crucial to the running of the program have been omitted for brevity.

Table : GameObject abstract class details

|  |  |  |
| --- | --- | --- |
| Responsibility | Type Details | Notes |
| \_position | protected field, Point2D | The position of the game object. |
| \_width | protected field, double | The width of the game object. |
| \_height | protected field, double | The height of the game object. |
| \_sprite | protected field, Bitmap | The sprite (image) of the game object. |
| \_zIndex | protected field, int | A number that controls how the game object will be drawn in relation to other objects. Objects with higher z-indices are drawn on top. |
| GameObject | public constructor, parameters (Point2D position, double width, double height, Bitmap sprite, int zIndex = 1) | Used to create a new game object. |
| Draw | public virtual method, parameter (DrawingOptions options), returns void | Draws the game object to the screen, using the provided DrawingOptions. |
| Update | public abstract method, parameter (uint fps), returns void | Updates the game object. |

Table 2: Collider class details

|  |  |  |
| --- | --- | --- |
| Responsibility | Type Details | Notes |
| \_gameObject | private field, GameObject | The game object associated with the collider. |
| \_baseColliderBox | private field, Quad | A rectangle representing the collider whose center is rooted at (0, 0). The Quad type is chosen over Rectangle because the collider can be rotated. |
| Collider | public constructor, parameters (GameObject gameObject, double rotation) | Used to create a new collider object. |
| IsCollidingWith | public method, parameter (Collider c), returns bool | Determines if the collider is colliding with another collider. |
| GetColliderBox | private method, no paremeters, returns Quad | Returns a new Quad after moving the base collider box to the position of the game object. |
| GameObject | public readonly property, returns GameObject | Used to retrieve the game object associated with the collider. |

Table 3: StaticObject abstract class details

|  |  |  |
| --- | --- | --- |
| Responsibility | Type Details | Notes |
| … StaticObject inherits from GameObject | | |
| StaticObject | public constructor, parameters (Point2D position, double width, double height, Bitmap sprite, int zIndex = 1) | Used to create a new static game object. |
| Update | public override method, parameter (uint fps), returns void | Updates the static game object. |

Table 4: DynamicObject abstract class details

|  |  |  |
| --- | --- | --- |
| Responsibility | Type Details | Notes |
| … DynamicObject inherits from GameObject | | |
| \_velocity | protected field, Vector2D | The velocity of the dynamic object. |
| DynamicObject | public constructor, parameters (Point2D position, double width, double height, Bitmap sprite, Vector2D velocity, int zIndex = 1) | Used to create a new dynamic game object. |
| Draw | public override method, parameter (DrawingOptions options), returns void | Draws the dynamic object, flipping the sprite across the Y axis depending on its FacingDirection. |
| Update | public override method, parameter (uint fps), returns void | Updates the dynamic game object, moving it by its velocity. |
| MoveTo | public method, parameter (Point2D point), returns void | Moves the object to the specified position. |
| MoveBy | public method, parameter (Vector2D displacement), returns void | Moves the object along the specified vector. |

Table 5: Projectile class details

|  |  |  |
| --- | --- | --- |
| Responsibility | Type Details | Notes |
| … Projectile inherits from DynamicObject and implements the ICollidable and IDestroyable interfaces | | |
| \_rotation | private field, double | The projectile’s rotation measured counter-clockwise from the vector (1, 0). Used to draw the projectile and set up its collider. |
| \_collider | private field, Collider | The Collider object associated with the projectile. |
| \_canDestroy | private field, bool | Whether the projectile can be destroyed or not. |
| \_projectileType | private field, ProjectileType | The type of the projectile. Can be either Friendly or Hostile. |
| \_damage | private field, int | The projectile’s damage. |
| Projectile | public constructor, parameters (Point2D position, double width, double height, Bitmap sprite, Vector2D initialVelocity, double targetSpeed, ProjectileType type, int damage) | Used to create a new projectile. |
| Draw | public override method, parameter (DrawingOptions options), returns void | Draws the projectile, rotating it by \_rotation. |
| Collide | public virtual method, parameter (Collider c), returns void | Defines the projectile’s behavior on colliding with an object. |
| Collider | public readonly property, returns Collider | Used to retrieve the projectile’s collider. |
| CanDestroy | public readonly property, returns bool | Used to determine if the projectile can be destroyed. |
| Destroy | public method, no parameters, returns void | Defines the projectile’s behavior when destroyed. |

Table 6: ICollidable interface details

|  |  |  |
| --- | --- | --- |
| Responsibility | Type Details | Notes |
| Collider | readonly property, returns Collider | Used to retrieve the collider (a hitbox) associated with a game object. |
| Collide | method, parameter (Collider collider), returns void | A method that defines the object’s behavior upon collision with another object. |

Table 7: IDestroyable interface details

|  |  |  |
| --- | --- | --- |
| Responsibility | Type Details | Notes |
| CanDestroy | readonly property, returns bool | Used to determine whether an object can be destroyed or not. Destroying means removing all references to an object. |
| Destroy | method, no parameters, returns void | Defines the object’s behavior or additional logic when it is destroyed. |

Table 8: Character abstract class details

|  |  |  |
| --- | --- | --- |
| Responsibility | Type Details | Notes |
| Character inherits from DynamicObject and ICollidable | | |
| \_health | protected field, int | The character’s current health. |
| \_maxHealth | protected field, int | The character’s maximum health. |
| \_collider | private field, Collider | The collider associated with the character. |
| Character | public constructor, parameters (Point2D position, double width, double height, Bitmap sprite, Vector2D initialVelocity, int maxHealth, int zIndex = 1) | Used to create a new character. |
| Collider | public readonly property, returns Collider | Used to retrieve the character’s collider. |
| Collide | public virtual method, parameters (Collider c), returns void | Defines the character’s behavior on colliding with an object. |
| Damage | public virtual method, parameters (int amount), returns void | Damages the character by a particular amount. |
| Heal | public virtual method, parameters (int amount), returns void | Heals the character by a particular amount. |
| IsDead | public method, no parameters, returns bool | Used to determine if the character is dead. |

Table 9: Player class details

|  |  |  |
| --- | --- | --- |
| Responsibility | Type Details | Notes |
| Player inherits from Character | | |
| \_weapon | private field, Weapon | The player’s equipped weapon. |
| \_spell | private field, Spell? | The player’s equipped spell, can be null. |
| \_potion | private field, HealthPotion | The player’s equipped health potion. |
| \_experience | private field, int | The player’s current experience. |
| \_level | private field, int | The player’s level. |
| Player | public constructor, parameters (Point2D position, Vector2D initialVelocity, int maxHealth) | Used to create a new player. |
| Halt | public method, returns void | Sets the player’s velocity to 0. |
| MoveAlong | public method, parameters (Vector2D direction) | Sets the player’s velocity to move along the specified direction. |
| Attack | public method, parameters (Point2D target) | Attacks a target at the specified location. |
| UseSpell | public method, no parameters, returns void | Casts the player’s currently equipped spell. |
| DrinkPotion | public method, no parameters, returns void | Drinks a health potion. |
| AddExperience | public method, parameter (int amount), returns void | Adds experience to the player. |
| TakeNewItem | public method, parameter (Item newItem), returns Item | Takes a new item such as a Weapon or Spell, returning the old one. |

Table 10: Enemy abstract class details

|  |  |  |
| --- | --- | --- |
| Responsibility | Type Details | Notes |
| Enemy inherits from Character and IDestroyable | | |
| \_experienceValue | protected field, int | The experience yielded by killing this enemy. |
| \_attackDamage | protected field, int | The enemy’s attack damage. |
| \_attackCooldown | protected field, double | The time in seconds between the enemy’s attacks. |
| Enemy | public constructor, parameters (Point2D position, double width, double height, Bitmap sprite, Vector2D initialVelocity, int maxHealth, int attackDamage, double attackCooldown, int experienceValue) | Used to create a new enemy. |
| Attack | protected abstract method, parameters (Player p) | Defines the enemy’s attack behavior. Must be implemented by derived classes. |
| Move | protected abstract method, parameters (Player p) | Defines the enemy’s movement behavior. Must be implemented by derived classes. |
| Update | public override method, parameters (uint fps), returns void | Calls the attack and movement methods of the enemy. |
| CanDestroy | public readonly property, returns bool | Used to determine if the enemy object can be destroyed. |
| Destroy | public method, no parameters, returns void | Gives the player experience when the enemy is killed. |

Table 11: Shrub class details

|  |  |  |
| --- | --- | --- |
| Responsibility | Type Details | Notes |
| Shrub inherits from Enemy | | |
| Shrub | public constructor, parameters (Point2D position, int floorLevel) | Used to create a new shrub enemy object. The parameter floorLevel is used to scale the strength of the shrub. |
| Attack | protected overide method, parameter (Player player), returns void | Defines the attack behavior of the shrub. |
| Move | protected override method, parameter (Player player), returns void | Defines the movement behavior of the shrub. |

Table 12: Wizard class details

|  |  |  |
| --- | --- | --- |
| Responsibility | Type Details | Notes |
| Wizard inherits from Enemy | | |
| Wizard | public constructor, parameters (Point2D position, int floorLevel) | Used to create a new wizard enemy object. The parameter floorLevel is used to scale the strength of the wizard. |
| Attack | protected overide method, parameter (Player player), returns void | Defines the attack behavior of the wizard. |
| Move | protected override method, parameter (Player player), returns void | Defines the movement behavior of the wizard. |

Table 13: Wall class details

|  |  |  |
| --- | --- | --- |
| Responsibility | Type Details | Notes |
| Wall inherits from StaticObject and ICollidable | | |
| \_collider | private field, Collider | The collider object associated with the wall. |
| Wall | public constructor, parameters Point2D position, double width, double height, Bitmap sprite | Used to create a new wall. |
| Collider | public readonly property, returns Collider | Used to retrieve \_collider. |
| Collide | public method, parameter Collider c | Defines the wall’s behavior on collision with another object. |

Table 14: Exit class details

|  |  |  |
| --- | --- | --- |
| Responsibility | Type Details | Notes |
| Exit inherits from StaticObject and ICollidable | | |
| \_collider | private field, Collider | The collider object associated with the exit. |
| \_sourceRoom, \_destinationRoom | private fields, Room | The source and destination rooms associated with the exit. |
| \_direction | private field, Direction | The direction of the exit. |
| Exit | public constructor, parameters Point2D position, double width, double height, Bitmap sprite, Direction direction, Room sourceRoom, Room destinationRoom | Used to create a new exit. |
| Collider | public readonly property, returns Collider | Used to retrieve \_collider. |
| Collide | public method, parameter Collider c | Defines the exit’s behavior on collision with another object. |

Table 15: Interactable abstract class details

|  |  |  |
| --- | --- | --- |
| Responsibility | Type Details | Notes |
| Interactable inherits from StaticObject | | |
| \_range | private field, double | The range within which the object can be interacted with. |
| Interactable | public constructor, parameters Point2D position, double width, double height, Bitmap sprite, double range | Used to create a new interactable object. |
| IsNearPlayer | public method, parameter Player p, returns bool | Determines whether the specified player is within range of the interactable. |
| IsHoveredOn | public method, parameter Point2D mousePosition, returns bool | Determines whether the mouse is on top of the interactable. |
| HandleInteraction | public abstract method, returns void | Defines the behavior of the interactable when it is clicked on. Must be implemented by derived classes. |

Table 16: Staircase class details

|  |  |  |
| --- | --- | --- |
| Responsibility | Type Details | Notes |
| Staircase inherits from Interactable | | |
| Staircase | public constructor, parameter Point2D position | Used to create a new staircase object. |
| HandleInteraction | public override method, returns void | Defines the behavior when the staircase is clicked on. |

Table 17: Chest class details

|  |  |  |
| --- | --- | --- |
| Responsibility | Type Details | Notes |
| Chest inherits from Weapon and implements ICollidable | | |
| \_collider | private field, Collider | The chest’s collider object. |
| \_items | private field, List<Item> | The list of items inside the chest. |
| Chest | public constructor, parameters (Point2D position, int floorLevel) | Used to create a new chest object. |
| GenerateChestContent | private static method, parameter (int floorLevel), returns List<Item> | Generates a list of items whose statistics scale with the floor level. |
| HandleInteraction | public override method, no parameters, returns void | Opens the current chest, changing the game state to OpenChest and displaying the chest content to the screen. |
| Collider | public readonly property, returns Collider | Used to retrieve the chest’s collider object. |
| Collide | public method, parameter (Collider c), returns void | Defines the behavior of the chest when colliding with another object. |
| GetItem | public method, parameter (int index), returns Item? | Retrieves an item at index “index” from the list of items. Returns null if the index is invalid. |
| AddItem | public method, parameter (Item item), returns void | Adds an item to the chest. |

Table 18: Item abstract class details

|  |  |  |
| --- | --- | --- |
| Responsibility | Type Details | Notes |
| \_name | protected field, string | The item’s name. |
| \_description | protected field, string | The item’s description |
| \_icon | protected field, Bitmap | The item’s display icon. |
| Item | public constructor, parameters (string name, string description, Bitmap icon) | Used to create a new Item object. |
| DrawItem | public method, parameters (double x, double y), returns void | Draws the item’s information onto the screen at location (x, y). |

Table 19: HealthPotion class details

|  |  |  |
| --- | --- | --- |
| Responsibility | Type Details | Notes |
| HealthPotion inherits from Item | | |
| \_charges | private field, int | The number of charges the potion has left. |
| HealthPotion | public constructor, parameter (int charges) | Used to create a new health potion object with the specified number of charges. |
| Drink | public method, parameter (Player p), returns void | Heals the player p and remove one charge from the potion. Only works if the number of charges is greater than 0. |
| GetCharges | public method, no parameters, returns int | Returns the number of charges. |
| AddCharges | public method, parameter (int charges), returns void | Adds charges to the potion. |

Table 20: Weapon abstract class details

|  |  |  |
| --- | --- | --- |
| Responsibility | Type Details | Notes |
| Weapon inherits from Item | | |
| \_damage | private field, int | The weapon’s damage |
| \_attackCooldown | private field, double | The weapon’s attack cooldown in seconds. |
| Weapon | public constructor, parameters (string name, string description, Bitmap icon, int damage, double attackCooldown) | Used to create a new weapon object. |
| ReadyForAttack | protected virtual method, no parameters, returns bool | Determines if the weapon is off cooldown for an attack. |
| IncurCooldown | protected virtual method, no parameters, returns void | Puts the weapon on cooldown after an attack. |
| Attack | public method, parameter (Point2D target), returns void | Checks if the weapon is available for attack. If yes, attacks the target at the specified location and incurs a cooldown. |
| PerformAttack | protected abstract method, parameter (Point2D target), returns void | Specifies the attack behavior, must be implemented by derived classes. |

Table 21: Bow class details

|  |  |  |
| --- | --- | --- |
| Responsibility | Type Details | Notes |
| Bow inherits from Weapon | | |
| Bow | public constructor, parameters int damage, double attackCooldown | Used to create a new bow object. |
| PerformAttack | protected override method, parameter (Point2D target), returns void | Specifies the bow’s attack behavior. |

Table 22: SpellTome class details

|  |  |  |
| --- | --- | --- |
| Responsibility | Type Details | Notes |
| SpellTome inherits from Weapon | | |
| Bow | public constructor, parameters int damage, double attackCooldown | Used to create a new spell tome object. |
| PerformAttack | protected override method, parameter (Point2D target), returns void | Specifies the spell tome’s attack behavior. |
| ReadyForAttack | protected override method, no parameters, returns bool | Determines whether the spell tome is ready for another attack. |

Table 23: Spell abstract class details

|  |  |  |
| --- | --- | --- |
| Responsibility | Type Details | Notes |
| Spell inherits from Item | | |
| \_cooldown | private field, double | The spell cooldown in seconds. |
| Spell | public constructor, parameters (string name, string description, Bitmap icon, double cooldown) | Used to create a new Spell object. |
| CastSpell | public method, no parameters, returns void | Checks if the spell is available to cast. If yes, casts the spell and incurs a cooldown. |
| ReadyToCast | protected virtual method, no parameters, returns bool | Determines if the spell is available to cast. |
| PerformCast | protected abstract method, no parameters, returns void | Specifies the spell behavior when cast. Must be implemented by derived classes. |
| IncurCooldown | protected virtual method, no parameters, returns void | Puts the spell on cooldown after a cast. |

Table 24: HealSpell class details

|  |  |  |
| --- | --- | --- |
| Responsibility | Type Details | Notes |
| HealSpell inherits from Spell | | |
| \_healPercentage | private field, double | The strength of the heal spell. |
| HealSpell | public constructor, parameters (double healPercentage, double cooldown) | Used to create a new heal spell object. |
| PerformCast | protected override method, no parameters, returns void | Heals the player by a percentage of their missing health. |

Table 25: LightningSpell class details

|  |  |  |
| --- | --- | --- |
| Responsibility | Type Details | Notes |
| LightningSpell inherits from Spell | | |
| \_damage | private field, int | The damage of the spell. |
| LightningSpell | public constructor, parameters (int damage, double cooldown) | Used to create a new lightning spell object. |
| PerformCast | protected override method, no parameters, returns void | Damages all enemies on the screen by the amount specified by \_damage. |

Table 26: Room class details

|  |  |  |
| --- | --- | --- |
| Responsibility | Type Details | Notes |
| \_gameObjects | private field, List<GameObject> | The list of game objects in the room. |
| Room | private constructor | Used to create a new Room object. Can only be used inside the Room class. |
| CreateRoom | public static method, parameters bool hasNorthExit, bool hasEastExit, bool hasSouthExit, bool hasWestExit, returns Room | Creates a normal room. Use this method to create a new room outside the Room class. |
| CreateEndRoom | public static method, parameters bool hasNorthExit, bool hasEastExit, bool hasSouthExit, bool hasWestExit, returns Room | Creates a room with a staircase that can be clicked on to enter a new floor. Use this method to create a new room outside the Room class. |
| IsClear | public method, return bool | Determines if a room is cleared of all enemies. |
| GameObjects | public readonly property, returns List<GameObject> | Used to retrieve the list of game objects in the room. |
| AddExit | public method, parameters Direction direction, Room destination | Adds an exit in the specified direction that leads to the specified room. |

Table 27: Floor class details

|  |  |  |
| --- | --- | --- |
| Responsibility | Type Details | Notes |
| \_rooms | private field, Array of Rooms | A two-dimensional array of rooms in the floor. |
| \_startRoom | private field, Room | The starting room of the floor. |
| Floor | private constructor | Can only be used inside the class to create a new floor. |
| CreateFloor | public static method, returns Floor | Used to create a new floor outside the Floor class. |
| DrawMinimap | public method, parameters double x, double y, Room currentRoom | Draws the minimap showing the floor layout. |
| StartRoom | public readonly property, returns Room | Used to retrieve the starting room. |

Table 28: Game class details

|  |  |  |
| --- | --- | --- |
| Responsibility | Type Details | Notes |
| CurrentGame | public static field, Game? | The current and only running game instance. |
| \_window | private field, Window | The game window. |
| \_state | private field, GameState | The state of the game. |
| \_floor | private field, Floor | The current floor object. |
| \_floorCounter | private field, int | The current floor number. |
| \_currentRoom | private field, Room | The current room the player is in. |
| \_objectsOnScreen | private field, List<GameObject> | The list of game objects currently on the screen. |
| \_player | private field, Player | The active player. |
| \_activeChest | private field, Chest? | The currently opened chest. Nullable. |
| Game | private constructor | Used to create a new Game instance from inside the class. |
| CreateGame | public static method, returns Game | Creates and returns a new Game instance if there is none, otherwise returns CurrentGame. |
| Run | public method, returns void | Runs the game loop, which consists of: handling inputs, updating the game logic, handling collisions, and drawing objects. |
| CleanUp | public method, returns void | Cleans up the resources loaded for the game. |
| LoadResources | private method, returns void | Loads the game resources. |
| HandleInputs | private method, returns void | Handles the user’s inputs. |
| Update | private method, returns void | Updates the game logic. |
| HandleCollisions | private method, returns void | Handles collisions between game objects. |
| Draw | private method, returns void | Draws game objects onto the screen. |
| AddGameObjectOnScreen | public method, parameter GameObject gameObject, returns void | Used to add a game object onto the screen (e.g., a projectile). |
| EnterRoom | public method, parameters Room room, Direction enterDirection, returns void | Enters the specified room from the specified direction. |
| CurrentPlayer | public readonly property, returns Player | Used to retrieve the active player. |
| EnterNewFloor | public method, returns void | Creates and enters a new floor, incrementing the floor number. |
| ResetGame | private method, returns void | Resets the game after the player has died. |
| OpenChest | public method, parameter (Chest chest), returns void | Changes the game state to OpenChest and sets the active chest to the specified chest. |
| CloseChest | private method, no parameters, returns void | Changes the game state to Playing and sets the active chest to null. |
| HandleChestInteraction | private method, no parameters, returns void | Called when the game state is OpenChest and the user left-clicks. It determines if the left click is on one of the chest items’ TAKE buttons, and what index of the item it is. It then takes the item from the chest and gives it to the player. |

Table 29: Direction enumeration details

|  |  |
| --- | --- |
| Value | Notes |
| North | Cardinal directions of exit gates. |
| East |
| South |
| West |

Table 30: GameState enumeration details

|  |  |
| --- | --- |
| Value | Notes |
| Playing | The player can press ESC to pause/unpause. |
| Paused |
| Lost | The game state is Lost when the player’s health reaches 0. |
| OpenChest | When the player opens a chest, the game state becomes OpenChest, pausing the game and revealing the chest’s contents. |

# Class Diagram

A diagram of a diagram

Description automatically generated

*Figure 2: The program’s class diagram. Details of the classes and interfaces are omitted to keep the diagram small and readable. See the classes’ details in the previous section. Each red zone shows the location of a design pattern used in the program.*

# Design Patterns

## Singleton

A screenshot of a computer program

Description automatically generated

*Figure 3: UML diagram for the Game class.*

The singleton design pattern is used to ensure one instance of a class is ever created and there is a global point of access to it. The class Game is a singleton in this program since only one game should ever be created throughout the program’s lifespan. In code, this is achieved by adding to the class a public static field named CurrentGame that takes the type of ***Game?***. The question mark indicates that this field can hold a null value. This is because before the first Game instance is created, CurrentGame should be null.

The constructor of Game must be made private to prevent outsiders from creating another Game instance with new. To get a Game object, they instead must call the public static CreateGame method. This method creates a new game from the private constructor, assigns it to CurrentGame, and returns it if CurrentGame is null. Otherwise, it returns CurrentGame.

## Strategy

A close-up of a computer code

Description automatically generated

*Figure 4: UML diagram for the Weapon class and its derived classes.*

The strategy pattern involves defining a group of related algorithms, putting them into separate classes, and using their objects interchangeably. In this program, the strategy pattern is used for the Weapon classes. Different weapons have different modes of attack (algorithms) but should be used interchangeably. The abstract class Weapon defines an Attack method, which serves as a common interface for the different concrete weapons. The Attack method calls ReadyForAttack, PerformAttack, and IncurCooldown, all of which can be overridden by concrete weapon classes to specify their exact behaviors, resulting in a different Attack behavior for each weapon. The Player maintains a reference to a Weapon and does not have to concern itself with the exact kind of weapon it is holding. To attack, the player merely calls the Attack method, which is common to all weapons.

## Template method

A diagram of a computer program

Description automatically generated

*Figure 5: UML diagram for the Weapon class and its derived classes, showing the use of the template method pattern.*

The template method pattern involves defining the skeleton of an algorithm in the base class but leaving the derived classes to override the specific steps. In this program, the template method pattern is used in the Weapon class. The abstract Weapon class defines the skeleton of the Attack method, which includes the steps in the order noted above. The concrete weapon classes can override the steps to specify their own behavior. For instance, SpellTome overrides the ReadyForAttack method to prevent the player from attacking if their health is below a certain threshold, as well as the PerformAttack method to specify its attack behavior.

# Improvements from D program

The HD version of this program adds more variety to the gameplay by introducing a new weapon, a spell mechanic, and healing potions. The player can find weapons, spells, and potions in chests that randomly spawn throughout the floors. Spells can be cast by pressing E, which triggers special effects depending on the spell type. For example, the Lightning Spell damages all enemies on the screen when cast. Healing potions can be drunk by pressing Q to heal the player.

The HD version also makes combat more challenging and rewarding by making enemies more powerful as the player progresses through floors. The player gains experience through defeating enemies and can level up for health and damage boosts.

A screenshot of a video game

Description automatically generated

*Figure 6: The game screen.*

# Sequence Diagrams

A black and white image of a chart

Description automatically generated

*Figure 7: The sequence diagram of the game loop. The Run method of the Game instance is called to initiate the game loop. The loop terminates when the user closes the game window.*

*A black and white text box

Description automatically generated*

*Figure 8: The structure diagram of the HandleCollisions method, which is called every frame to handle collision logic. \_objectOnScreen is a list of all game objects on the screen.*

*A diagram of a project

Description automatically generated*

*Figure 9: The structure diagram of the HandleChestInteraction method. This method is called every time a chest is open and the user left-clicks. If the user clicks on one of the items’ TAKE buttons, that item is given to the player. The player will return an old item of the same type (if applicable) and that item is added to the chest.*