University of Lethbridge

Practical Software Development

CPSC 2720

Wrexorcist

Brayden Carlson
brayden.carlson@uleth.ca

Angela De Sousa Costa
angela.desousacosta@uleth.ca

Rocky Au
rocky.au@uleth.ca

Juhyoung Park
juhyoung.park@uleth.ca

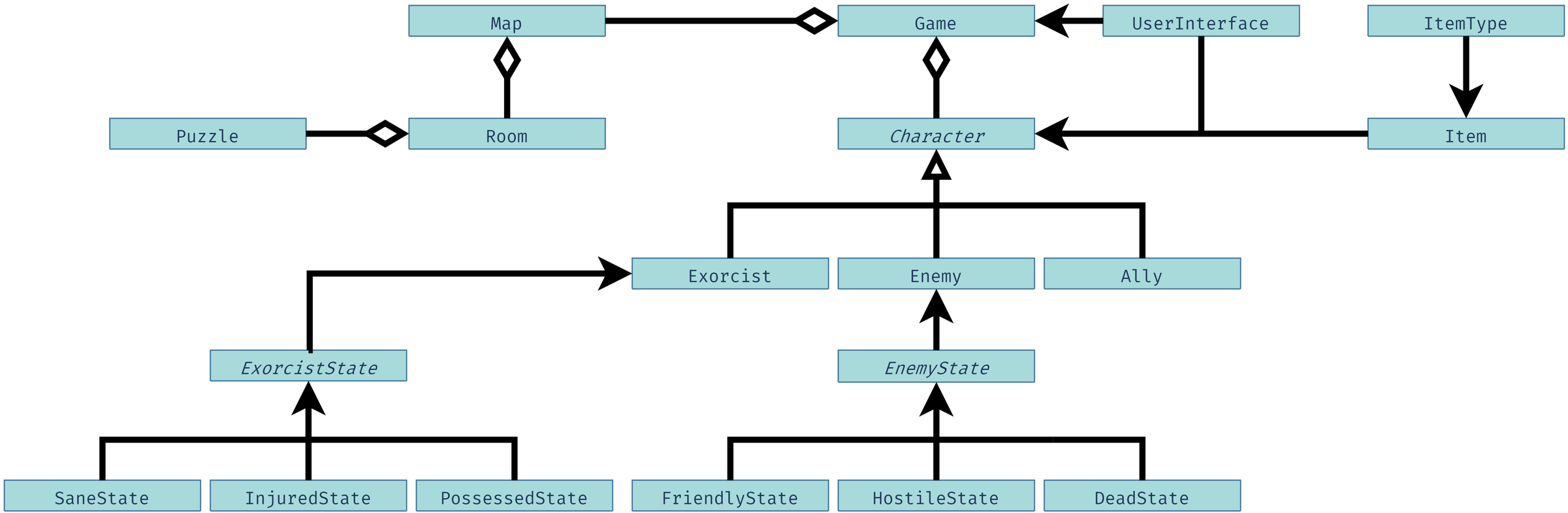
Introduction to The Wrexorcist

In this text-based adventure game you will take the role of a young exorcist in training. One night on your travels, you fall asleep in the woods and wake up in a strange world. A mysterious figure approaches you and informs you that there are three trapped souls who are keeping you here and it is your duty to release them. As you continue your adventure, you realize that there is more to this world than you think, and it is your duty to investigate what is truly going on. The actions that you take will have an impact on whether or not you escape.

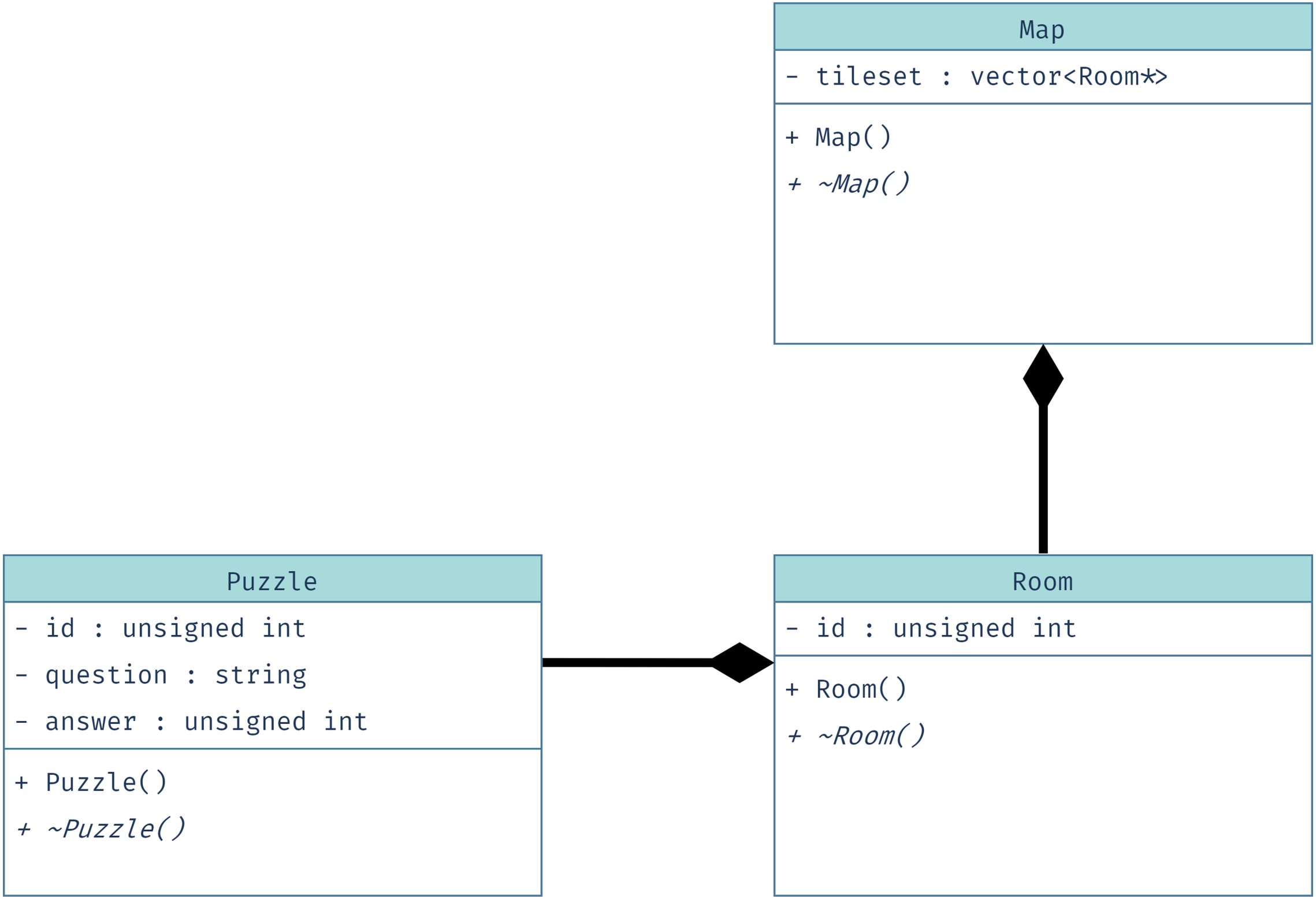
The player will be able to explore this mysterious world and interact with their environment. The player will examine the environment and enemies to discover clues and items to aid them on their adventure. There are also puzzles throughout the environment for unlockable items in order to get the best ending. Other locations include areas for the player to rest at and others that provide useful functions. There are also enemies present in this world, however, the player does not always need to fight the enemy, and in fact, may be able to befriend some of them. The player will need to figure out how to obtain the right items to appease some of these enemies (namely, the trapped souls). The player must be careful though, as some items may make the trapped souls permanently hostile. The player will also be able to find equipment and consumables should they desire to take the more combative route and the game will feature the possibility to fight the enemies.

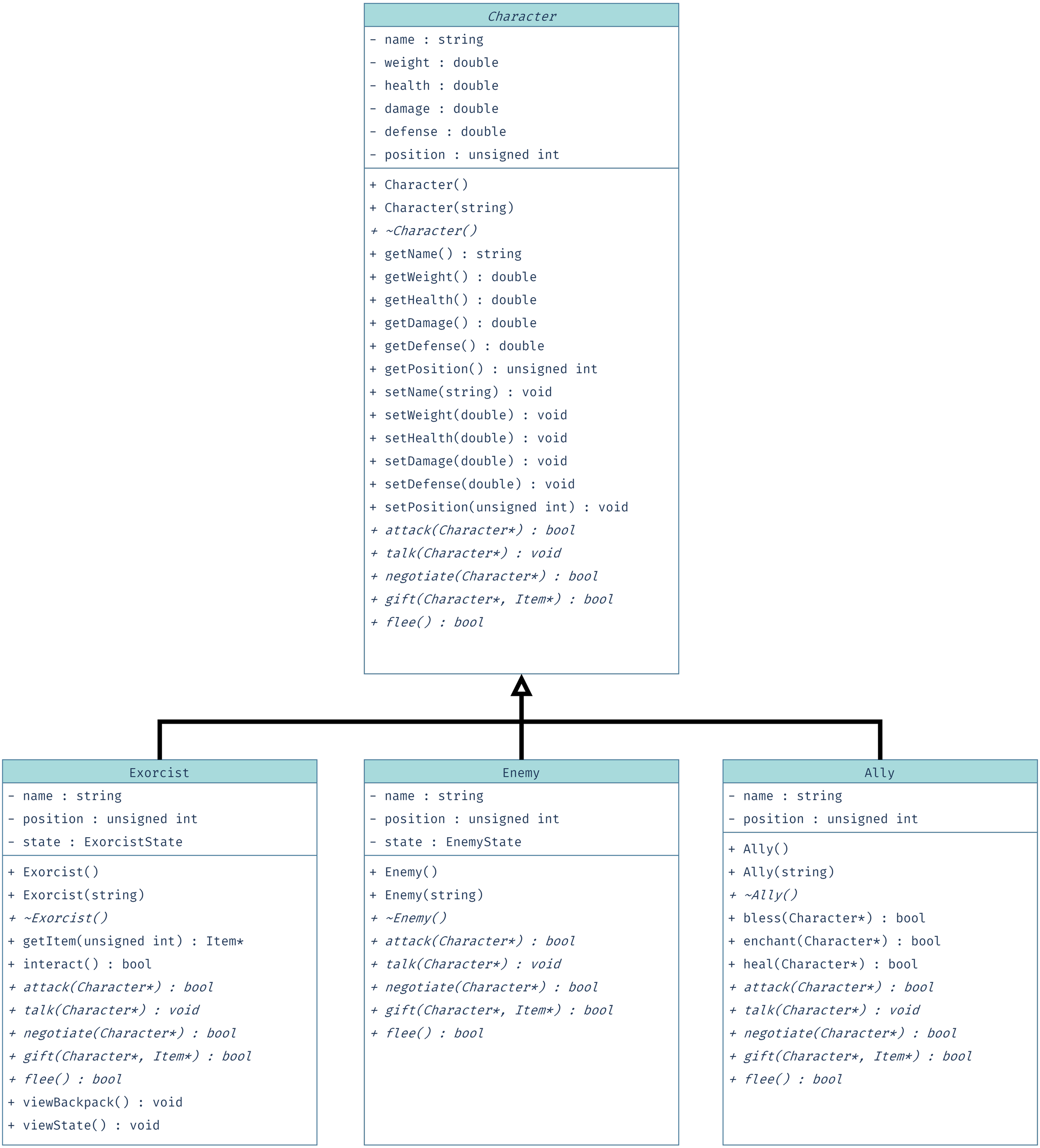
The environment will be initialized as a 4 x 4 area to explore, however it will be randomly generated. There will still be a natural progression to the exploration and enemies encountered, however the environment will not always be the same with each play through. The areas will be themed after the specific trapped boss in that area. This adds to the replayability of the game. There will be a variety of rooms throughout the game. Some rooms will contain generic enemies and items. There will also be utility rooms, such as for healing and teleporting. There will also be rooms with puzzles and traps that the player must solve to navigate. The items the players need will also be found in the rooms. Items include consumable items (health potions), equipment, and key items to exorcise the trapped souls.

The player completes the game if they are able to fulfill the conditions to exit the world. The final boss is the mysterious figure at the start of the game. The player must defeat the final boss before exiting the mysterious world. The game is only beatable if they do not kill every trapped soul. If they do, the final boss will become undefeatable, and the player will eventually run out of health. The player may also run out of health via combat or failing certain puzzles/traps enough times. If the player kills some of the trapped souls, but frees the others, the game goes towards a neutral end. If the player releases all of the souls, the player achieves the good ending, provided they defeat the final boss. The trapped souls can be set free by reducing their health to a certain amount and then using an item from their past on them. This will pacify the trapped soul and they will move on to the after life.

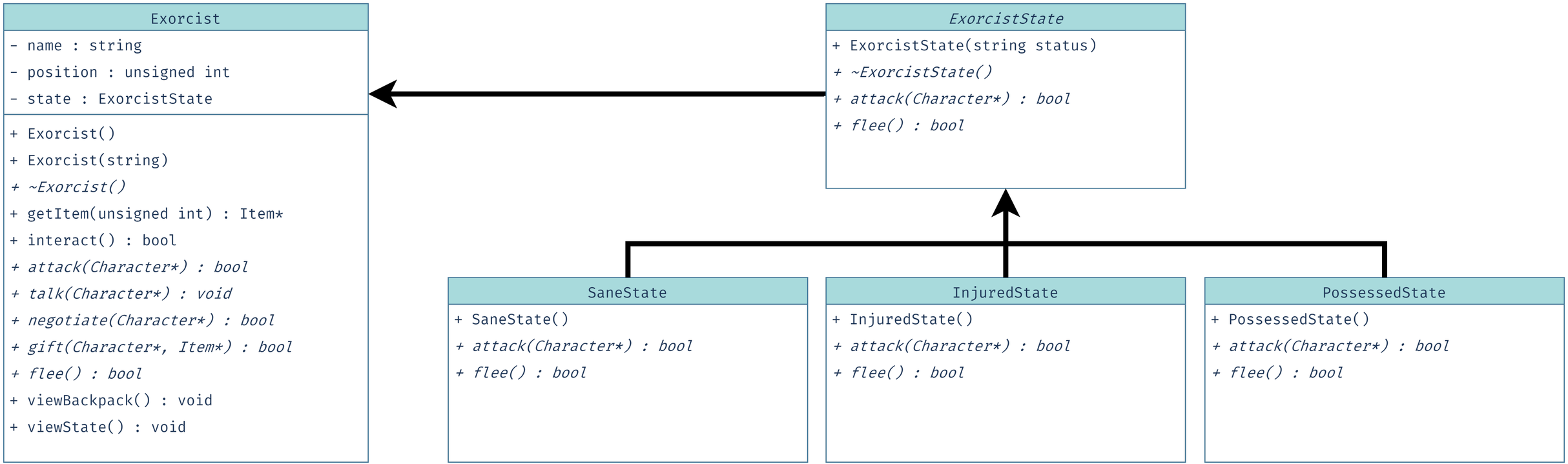

Map
Game
UserInterface
ItemType
Puzzle
Room
Character
Item
Exorcist
Enemy
Ally
ExorcistState
EnemyState
SaneState
InjuredState
PossessedState
FriendlyState
HostileState
DeadState

Game
- isRunning : bool
+ Game() *+ ~Game()* + play() : void + stop() : void

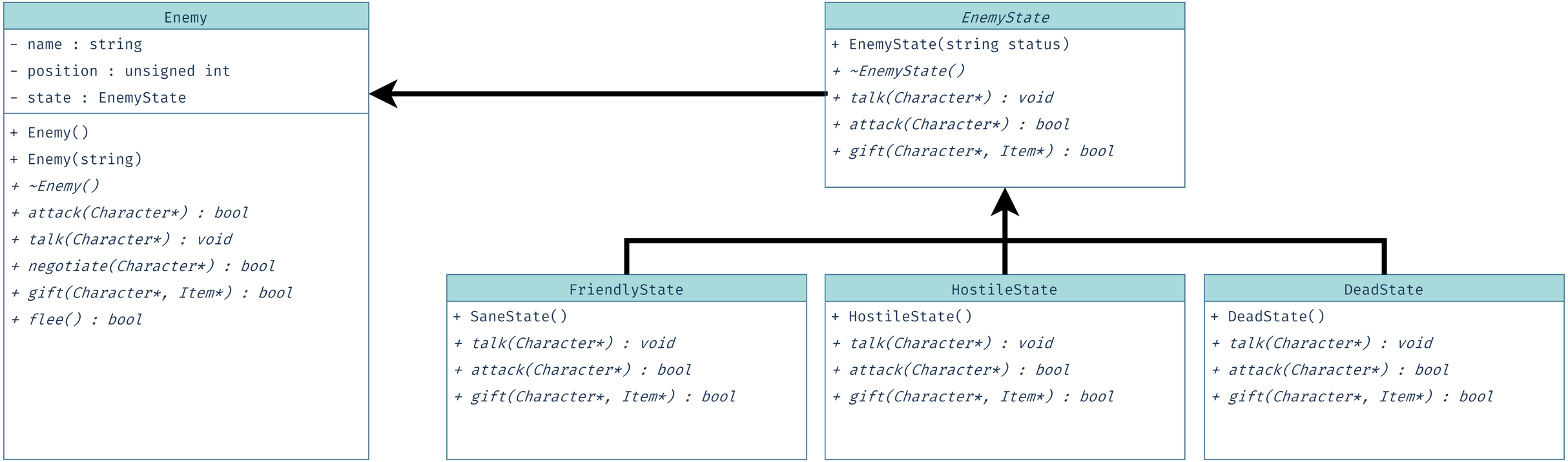

Map
- tileset : vector<Room*>
+ Map()
+ ~Map()
Puzzle
- id : unsigned int
- question : string
- answer : unsigned int
+ Puzzle()
+ ~Puzzle()
Room
- id : unsigned int
+ Room()
+ ~Room()

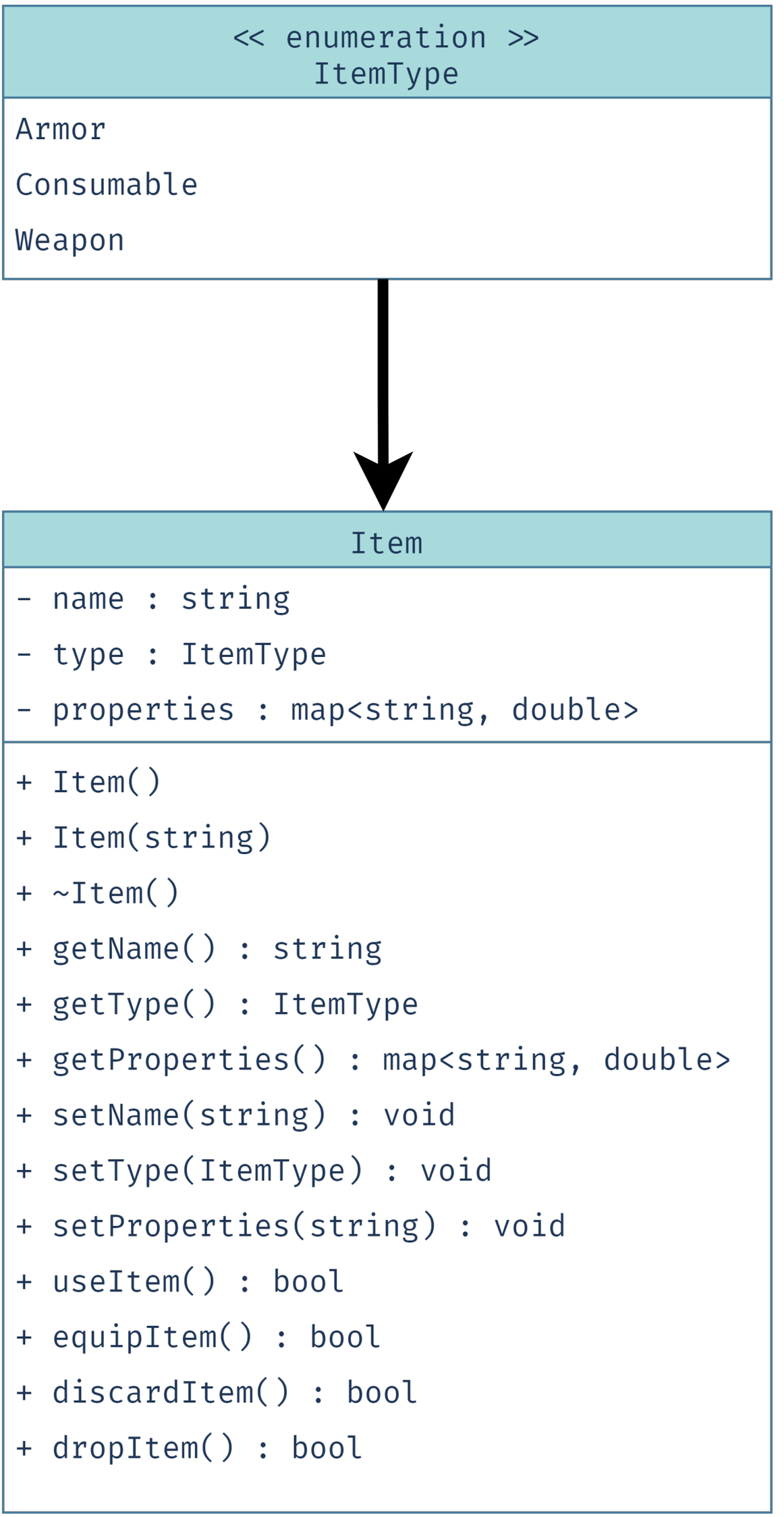

Character
- name : string
- weight : double
- health : double
- damage : double
- defense : double
- position : unsigned int
+ Character()
+ Character(string)
+ ~Character()
+ getName() : string
+ getWeight() : double
+ getHealth() : double
+ getDamage() : double
+ getDefense() : double
+ getPosition() : unsigned int
+ setName(string) : void
+ setWeight(double) : void
+ setHealth(double) : void
+ setDamage(double) : void
+ setDefense(double) : void
+ setPosition(unsigned int) : void
+ attack(Character*) : bool
+ talk(Character*) : void
+ negotiate(Character*) : bool
+ gift(Character*, Item*) : bool
+ flee() : bool
Exorcist
- name : string
- position : unsigned int
- state : ExorcistState
+ Exorcist()
+ Exorcist(string)
+ ~Exorcist()
+ getItem(unsigned int) : Item*
+ interact() : bool
+ attack(Character*) : bool
+ talk(Character*) : void
+ negotiate(Character*) : bool
+ gift(Character*, Item*) : bool
+ flee() : bool
+ viewBackpack() : void
+ viewState() : void
Enemy
- name : string
- position : unsigned int
- state : EnemyState
+ Enemy()
+ Enemy(string)
+ ~Enemy()
+ attack(Character*) : bool
+ talk(Character*) : void
+ negotiate(Character*) : bool
+ gift(Character*, Item*) : bool
+ flee() : bool
Ally
- name : string
- position : unsigned int
+ Ally()
+ Ally(string)
+ ~Ally()
+ bless(Character*) : bool
+ enchant(Character*) : bool
+ heal(Character*) : bool
+ attack(Character*) : bool
+ talk(Character*) : void
+ negotiate(Character*) : bool
+ gift(Character*, Item*) : bool
+ flee() : bool

State Pattern

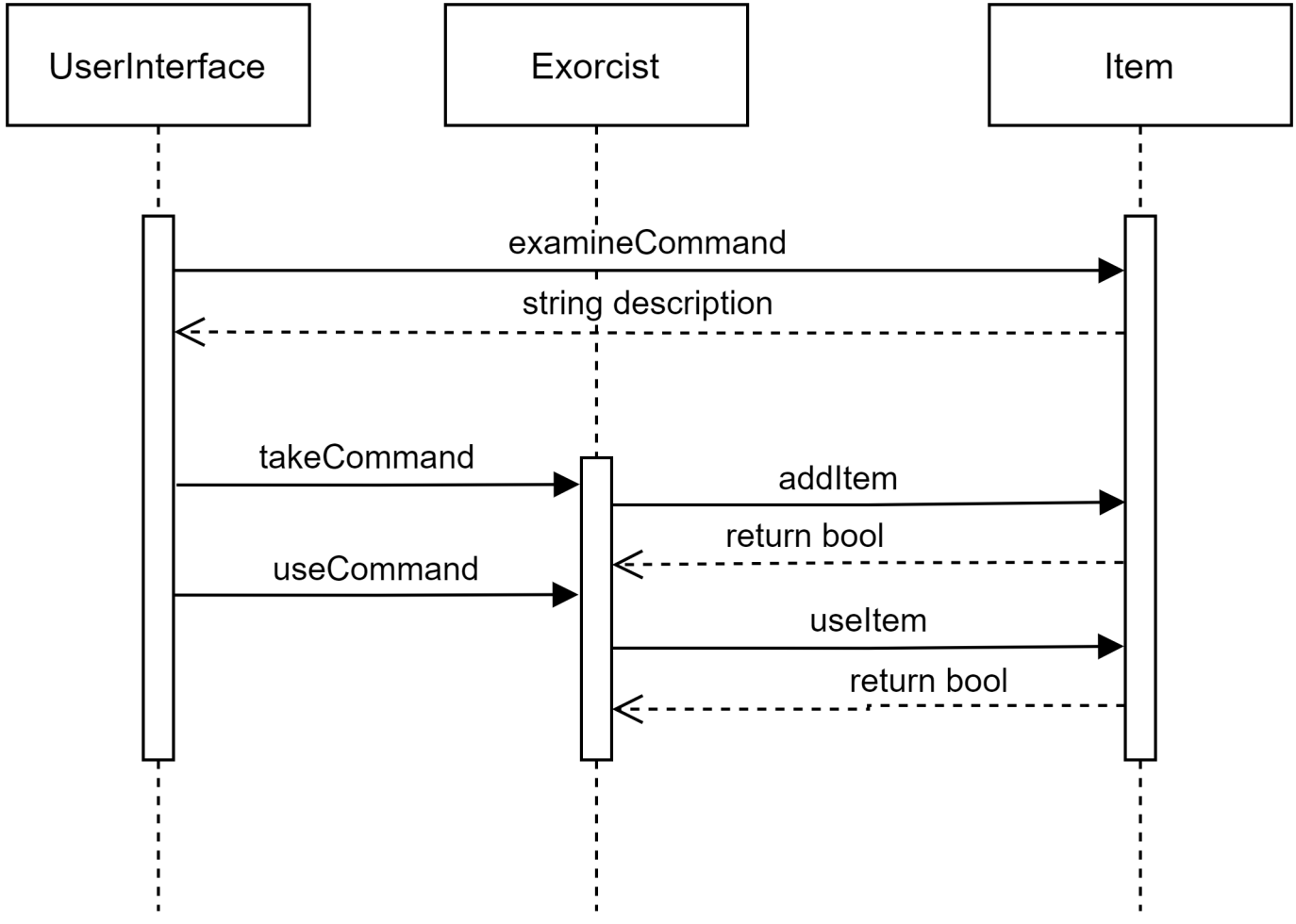


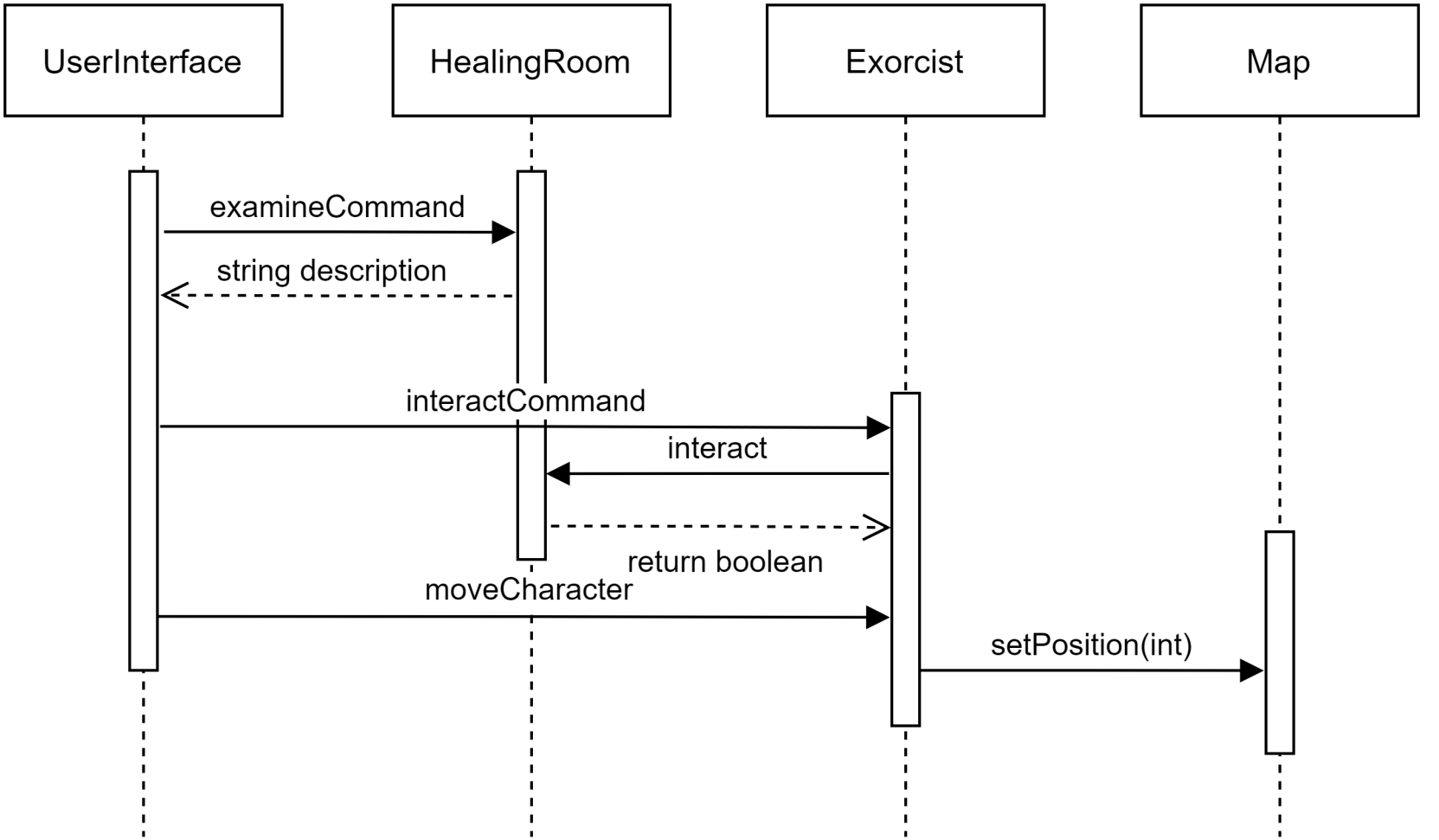
State Pattern

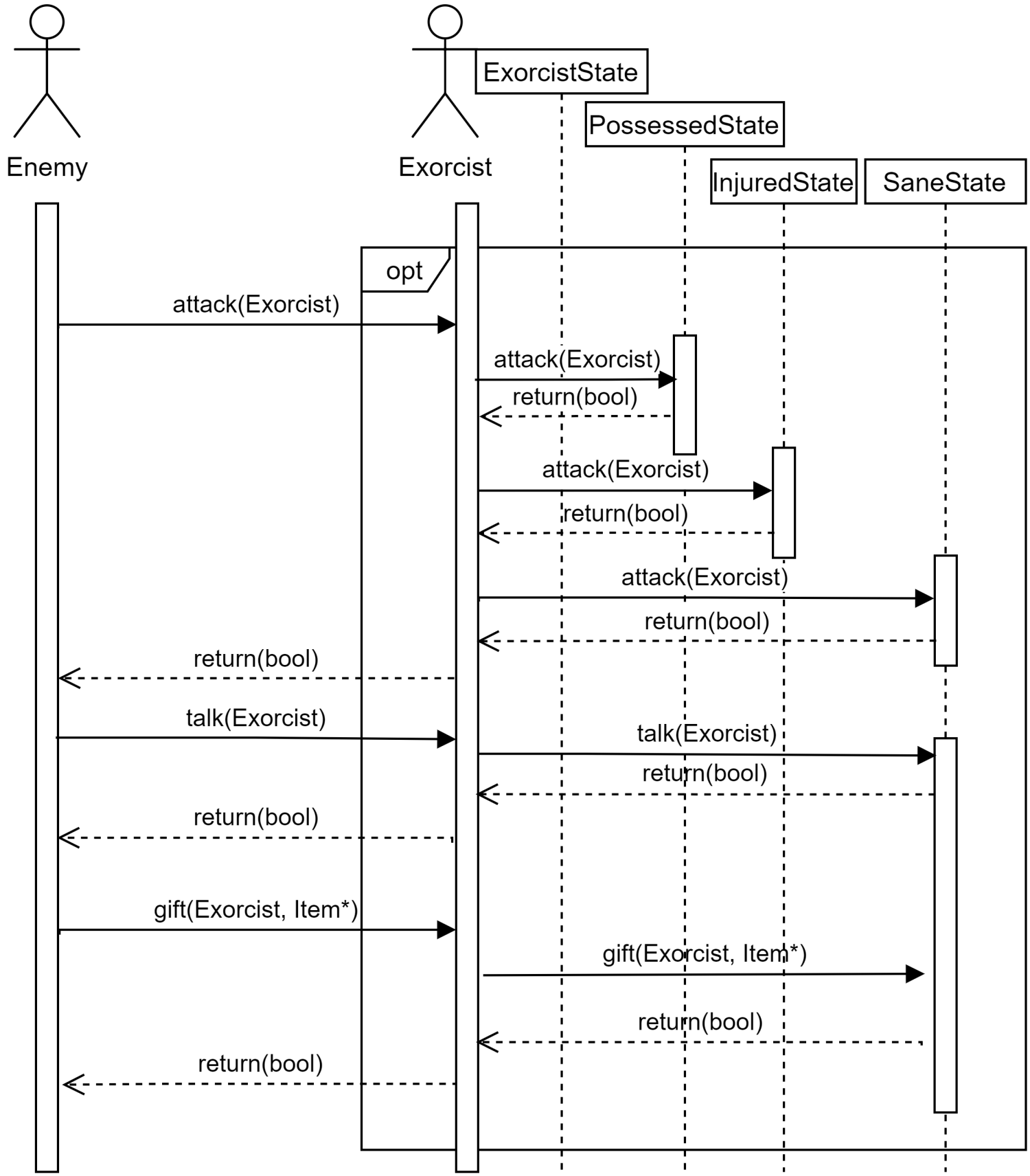


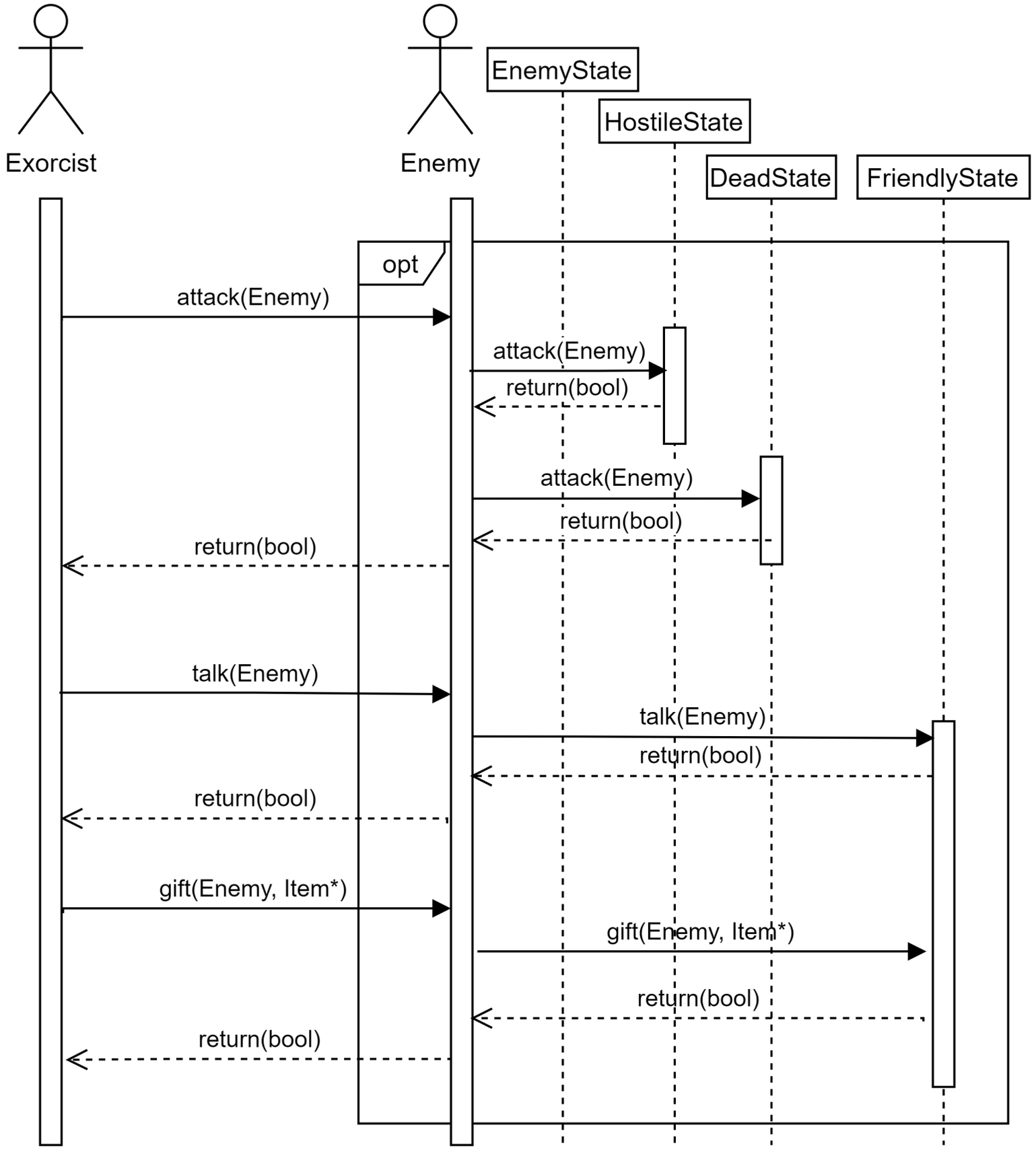

<< enumeration >>
ItemType
Armor
Consumable
Weapon
Item
- name : string
- type : ItemType
- properties : map<string, double>
+ Item()
+ Item(string)
+ ~Item()
+ getName() : string
+ getType() : ItemType
+ getProperties() : map<string, double>
+ setName(string) : void
+ setType(ItemType) : void
+ setProperties(string) : void
+ useItem() : bool
+ equipItem() : bool
+ discardItem() : bool
+ dropItem() : bool

```
UserInterface
+ help() : void
+ input(string) : void
+ examineCommand(string) : void
+ takeCommand() : void
+ talkCommand() : void
+ attackCommand() : void
+ fleeCommand() : void
```


UserInterface
Exorcist
Item
examineCommand
string description
takeCommand
addItem
return bool
useCommand
useItem
return bool


UserInterface
HealingRoom
Exorcist
Map
examineCommand
string description
interactCommand
interact
return boolean
moveCharacter
setPosition(int)


Enemy
Exorcist
ExorcistState
PossessedState
InjuredState
SaneState
opt
attack(Exorcist)
attack(Exorcist)
return(bool)
attack(Exorcist)
return(bool)
attack(Exorcist)
return(bool)
return(bool)
talk(Exorcist)
talk(Exorcist)
return(bool)
return(bool)
gift(Exorcist, Item*)
gift(Exorcist, Item*)
return(bool)
return(bool)


Exorcist
Enemy
EnemyState
HostileState
DeadState
FriendlyState
opt
attack(Enemy)
attack(Enemy)
return(bool)
attack(Enemy)
return(bool)
return(bool)
talk(Enemy)
talk(Enemy)
return(bool)
return(bool)
gift(Enemy, Item*)
gift(Enemy, Item*)
return(bool)
return(bool)

Game

A game session can be created, played and stopped, depending on the user's input.

The default constructor for Game.

```
Game()
```

The destructor for Game.

```
virtual ~Game()
```

Play the game

```
void play();
```

Stop the game

```
void stop();
```

Map

A container for each room. The map will be used to indicate a character's position in the world.

The default constructor for Map.

```
Map()
```

The destructor for Map.

```
virtual ~Map()
```

Room

The structure of a room. A room may contain trap(s), item(s), enemies or allies. A room and it's content will be dynamically generated.

The default constructor for Room.

```
Room()
```

The destructor for Room.

```
virtual ~Room()
```

Puzzle

A puzzle for the player to solve. A correct answer could result in a reward or furtherprogress, and an incorrect answer could result in losing health.

The default constructor for Puzzle.

```
Puzzle()
```

The destructor for Puzzle.

```
virtual ~Puzzle()
```

Character

An abstract base class for Exorcist, Enemy and Ally to inherit from.

The default constructor for Character.

```
Character()
```

A constructor for the Character class.
n is the name of the character to be created.

```
Character(std::string n)
```

The destructor for Character.

```
virtual ~Character()
```

Get the name of the character.
It returns the name of the character as a string.

```
std::string getName()
```

Get the weight of the character.
It returns the weight of the character as a double.

```
double getWeight()
```

Get the current health of the character.
It returns the current health of the character as a double.

```
double getHealth()
```

Get the amount of damage the character can do.
It returns the amount of damage the character can do as a double.

```
double getDamage()
```

Get the defensive modifier for a character.
It returns the defensive modifier of a character as a double.

```
double getDefense()
```

Get the position of the character in the map.
It returns the position of the character as an unsigned int.

```
unsigned int getPosition()
```

Set the name of the character.
n is the name to be set for the character.

```
void setName(std::string n)
```

Set the weight of the character.
w is the weight to be set for the character.

```
void setWeight(double w)
```

Set the health of the character.
h is the health to be set for the character.

```
void setHealth(double h)
```

Set the damage of the character.
d is the damage to be set for the character.

```
void setDamage(double d)
```

Set the defensive modifier of the character.
d is the defensive modifier to be set for the character.

```
void setDefense(double d)
```

Set the position of the character within the map.
p is the position to be set for the character.

```
void setPosition(double p)
```

Set the position of the character within the map.
p is the position to be set for the character.

```
void setPosition(double p)
```

Attack a character, friend or foe.
c is the pointer to a Character to attack

```
bool attack(Character* c)
```

Talk to a character, friend or foe.
c is the pointer to a Character to talk to

```
void talk(Character* c)
```

Negotiate with a character, friend or foe.
c is the pointer to a Character to talk to

```
bool negotiate(Character* c)
```

Gift a character an item, friend or foe.
c is the pointer to a Character to gift
i is the pointer to an Item to gift

```
bool gift(Character* c, Item* i)
```

Flee from an enemy if in combat.

```
bool flee()
```

Exorcist

A concrete class derived from the Character base class. The Exorcist is the main character, and consists of three states that will determine how likely an action will succeed and the amount of stat bonuses applied.

The default constructor for Exorcist.

```
Exorcist()
```

A constructor for the Exorcist class.
n is the name of the Exorcist to be created.

```
Exorcist(std::string n)
```

The destructor for Exorcist.

```
virtual ~Exorcist()
```

Get an item from the Exorcist's backpack.
i is the index of the item.
It returns a pointer to the item.

```
Item* getItem(unsigned int i)
```

The Exorcist attempts to interact with the environment.
It returns true if there is an interaction to be had, false otherwise.

```
bool interact()
```

The Exorcist attempts to attack a character, friend or foe. This method is affected by ExorcistState.
c is the pointer to a Character to attack.
It returns true if the attack was successful, false otherwise.

```
bool attack(Character* c)
```

The Exorcist attempts to talk to a character, friend or foe.
c is the pointer to a Character to talk to.
It returns true if able to talk to the character, false otherwise.

```
void talk(Character* c)
```

The Exorcist attempts to negotiate with a character, friend or foe.
c the pointer to a Character to negotiate with.
It returns true if the negotiation was successful, false otherwise.

```
bool negotiate(Character* c)
```

The Exorcist attempts to gift a character an item, friend or foe.
c the pointer to a Character to gift
i the pointer to an Item to gift
It returns true if gifting was successful, false otherwise.

```
bool gift(Character* c, Item* i)
```

The Exorcist attempts to flee from combat with an enemy.
This method is affected by ExorcistState.
It returns true if able to flee, false otherwise.

```
bool flee()
```

View the content of the Exorcist's backpack

```
void viewBackpack()
```

View the state of the Exorcist

```
void viewState()
```

Enemy

A concrete class derived from the Character base class. An enemy could be generic or a boss, and will be spawned in a hostile state. An item or defeat could change it's state.

The default constructor for Enemy.

```
Enemy()
```

A constructor for the Enemy class.
n is the name of the Enemy to be created.

```
Enemy(std::string)
```

The destructor for Enemy.

```
virtual ~Enemy()
```

The enemy attempts to attack a character. This method is affected by EnemyState.
c is the pointer to a Character to attack.
It returns true if the attack was successful, false otherwise.

```
bool attack(Character* c)
```

The enemy attempts to talk to a character. This method is affected by EnemyState.
c the pointer to a Character to talk to.
It returns true if able to talk to the character, false otherwise.

```
void talk(Character* c)
```

The enemy attempts to negotiate with a character.
c the pointer to a Character to negotiate with.
It returns true if the negotiation was successful, false otherwise.

```
bool negotiate(Character* c)
```

The enemy attempts to gift a character an item. This method is affected by EnemyState.
c the pointer to a Character to gift.
i the pointer to an Item to gift.
It returns true if gifting was successful, false otherwise.

```
bool gift(Character* c, Item* i)
```

The enemy attempts to flee from combat.
It returns true if able to flee, false otherwise.

```
bool flee()
```

Ally

A concrete class derived from the Character base class.
A friendly companion to assist the Exorcist in the game.

The default constructor for Ally.

```
Ally()
```

A constructor for the Ally class.
n is the name of the Ally to be created.

```
Ally(std::string)
```

The destructor for Ally.

```
virtual ~Ally()
```

The ally attempts to bless a character.
c is the pointer to a Character to bless.
It returns true if the blessing was successful, false otherwise.

```
bool bless(Character* c)
```

The ally attempts to enchant a character.
c is the pointer to a Character to enchant.
It returns true if the enchantment was successful, false otherwise.

```
bool enchant(Character* c)
```

The ally attempts to heal a character.
c is the pointer to a Character to heal.
It returns true if the healing was successful, false otherwise.

```
bool heal(Character* c)
```

The ally attempts to attack a character.
c is the pointer to a Character to attack.
It returns true if the attack was successful, false otherwise.

```
bool attack(Character* c)
```

The ally attempts to talk to a character.
c is the pointer to a Character to talk to.
it returns true if able to talk to the character, false otherwise.

```
void talk(Character* c)
```

The ally attempts to negotiate with a character.
c is the pointer to a Character to negotiate with.
It returns true if the negotiation was successful, false otherwise.

```
bool negotiate(Character* c)
```

The ally attempts to gift a character an item.
c is the pointer to a Character to gift
i is the pointer to an Item to gift
It returns true if gifting was successful, false otherwise.

```
bool gift(Character* c, Item* i)
```

The ally attempts to flee from combat.
It returns true if able to flee, false otherwise.

```
bool flee()
```

ExorcistState

ExorcistState is used to control various states of the Exorcist, depending on the context.

The default constructor for the ExorcistState class.
s is the name of the state.

```
ExorcistState(std::string s)
```

The destructor for ExorcistState.

```
virtual ~ExorcistState()
```

The Exorcist attempts to attack a character, depending
on the context and state of the Exorcist.
c is the pointer to a Character to attack.
It returns true if the attack was successful, false otherwise.

```
bool attack(Character* c)
```

The Exorcist attempts to flee from combat with an enemy, depending on the context and state of the Exorcist.
It returns true if able to flee, false otherwise.

```
bool flee()
```

SaneState

SaneState is a state that indicates that the Exorcist is in a normal state, which is the default. As the game progresses, the Exorcist may lose their sanity.

The default constructor for the SaneState class.

```
SaneState()
```

The Exorcist attempts to attack a character in the Sane state; the attempt will often be successful in this state.
c is the pointer to a Character to attack.
It returns true if the attack was successful, false otherwise.

```
bool attack(Character* c)
```

The Exorcist attempts to flee from an enemy while in the Sane state; the attempt will often be successful in this state.
It returns true if able to flee, false otherwise.

```
bool flee()
```

InjuredState

InjuredState is a state that indicates that the Exorcist is injured from running into a trap, answering a puzzle incorrectly or engaging in combat with enemies.

The default constructor for the InjuredState class.

```
InjuredState()
```

The Exorcist attempts to attack a character in the Injured state; the attempt will sometimes be successful in this state.
c is the pointer to a Character to attack.
It returns true if the attack was successful, false otherwise.

```
bool attack(Character* c)
```

The Exorcist attempts to flee from an enemy while in the Injured state; the attempt will sometimes be successful in this state.
It returns true if able to flee, false otherwise.

```
bool flee()
```

PossessedState

PossessedState is a state that indicates that the Exorcist is possessed and will require an item or blessing from an ally to go back to a Sane state. The game will be harder, and lower the Exorcist's stats.

The default constructor for the PossessedState class.

```
PossessedState()
```

The Exorcist attempts to attack a character in the Possessed state; the attempt will often fail in this state.
c is the pointer to a Character to attack.
It returns true if the attack was successful, false otherwise.

```
bool attack(Character* c)
```

The Exorcist attempts to flee from an enemy while in the Possessed state; the attempt will often be fail in this state.
It returns true if able to flee, false otherwise.

```
bool flee()
```

EnemyState

EnemyState is used to control various states of the Enemy, depending on the context.

The default constructor for the EnemyState class.
s is the name of the state.

```
EnemyState(std::string s)
```

The destructor for EnemyState.

```
virtual ~EnemyState()
```

The enemy attempts to talk to a character, depending on the context and state of the Enemy.
c is the pointer to a Character to talk to.
It returns true if able to talk to the character, false otherwise.

```
bool talk(Character* c)
```

The enemy attempts to attack a character, depending on the context and state of the Enemy.
c the pointer to a Character to attack.
It returns true if the attack was successful, false otherwise.

```
bool attack(Character* c)
```

The enemy attempts to flee from combat depending on the context and state of the Enemy.
It returns true if able to flee, false otherwise.

```
bool flee()
```

FriendlyState

FriendlyState is a state that indicates that the enemy is friendly, and will no longer attack the Exorcist, depending on context or item received.

The default constructor for the FriendlyState class.

```
FriendlyState()
```

The enemy will talk to a character in a friendly manner.
c is the pointer to a Character to talk to.
It returns true if able to talk to the character, false otherwise.

```
bool talk(Character* c)
```

The enemy will not attack a character in the Friendly state; the attempt will fail.
c the pointer to a Character to attack.
It returns false.

```
bool attack(Character* c)
```

The enemy will flee from combat if state is changed to friendly; the attempt will be successful in this state.
It returns true.

```
bool flee()
```

HostileState

HostileState is a state that indicates that the enemy is hostile, and will attack and engage in combat with the Exorcist, depending on context or until the enemy is defeated.

The default constructor for the HostileState class.

```
HostileState()
```

The enemy will talk to a character in a hostile manner.
c is the pointer to a Character to talk to.
It returns true if able to talk to the character, false otherwise.

```
bool talk(Character* c)
```

The enemy attempts to attack a character in the Hostile state; the attempt will often be successful in this state.
c is the pointer to a Character to attack.
It returns true if the attack was successful, false otherwise.

```
bool attack(Character* c)
```

The enemy attempts to flee from combat while in the Hostile state; the attempt will sometimes be successful in this state.
It returns true if able to flee, false otherwise.

```
bool flee()
```

DeadState

DeadState is a state that indicates that the enemy is dead through combat or an item/spell, and will no longer attack or converse with a character.

The default constructor for the DeadState class.

```
DeadState()
```

The enemy may or may not speak in this state; the attempt may result in muttered words.
c the pointer to a Character to talk to.
It returns true if able to talk to the character, false otherwise.

```
bool talk(Character* c)
```

The enemy is unable to attack while dead; the attempt will fail in this state.
c the pointer to a Character to attack.
It returns false.

```
bool attack(Character* c)
```

The enemy is unable to flee while dead; the attempt will fail in this state.
It returns false.

```
bool flee()
```

ItemType

The classification of the type of items.

```
enum class ItemType {
    ARMOR,
    CONSUMABLE,
    WEAPON,
};
```

Item

The structure and functionality of an item that can be used, modified or consumed.

The default constructor for the Item class.

```
Item()
```

A constructor for the Item class.
s is the name of the item.

```
Item(std::string s)
```

The destructor for Item.

```
virtual ~Item()
```

Get the name of the item.
It returns the name of the item as a string.

```
std::string getName()
```

Get the type of the item.
It returns the type of the item as an enumeration.

```
ItemType getType()
```

Get the properties of the item.
It returns the properties of the item as a map¡string, double¿.

```
map<string, double> getProperties()
```

Set the name of the item.
n is the name to be set for the item.

```
void setName(std::string n)
```

Set the type of the item.
t is the type to be set for the item

```
void setType(ItemType t)
```

Set the properties of the item.
p is the properties to be set for the item

```
void setProperties(std::string p)
```

A character attempts to use an item.
It returns true if item was successfully used, false otherwise.

```
bool useItem()
```

A character attempts to equip an item.
It returns true if item was successfully equipped, false otherwise.

```
bool equipItem()
```

A character attempts to discard an item.
It returns true if item was successfully discarded, false otherwise.

```
bool discardItem()
```

A character attempts to drop an item in the room.
It returns true if item was successfully dropped, false otherwise.

```
bool dropItem()
```

UserInterface

The user interface will manager user input to assist the player in playing the game.

Display all available help actions.

```
void help()
```

Ask for user input and accept valid input.
s is the input to parse.

```
void input(std::string s)
```

Display a description of the object or room the user examines.
e is the object or room that the user will examine.

```
void examineCommand(std::string e)
```

Invoke a sequence to add an item to the player's backpack.

```
void takeCommand()
```

Display the content of a player's attempt to talk to a character.

```
void talkCommand()
```

Invoke an attack on a character.

```
void attackCommand()
```

Invoke an attempt to flee from combat.

```
void fleeCommand()
```