


Assignment II Pair Blog - ABBA

Task 1) Code Analysis and Refactoring 🛠️

a) From DRY to Design Patterns

 Links to your merge requests

https://nw-syd-gitlab.cseunsw.tech/COMP2511/23T2/teams/H11A_ABBA/assignment-ii/-/merge_requests/1

i. Look inside `src/main/java/dungeonmania/entities/enemies`. Where can you notice an instance of repeated code? Note down the particular offending lines/methods/fields.

<Answer>

- In the `ZombieToast` and `Mercenary` `move()` methods, particularly the blocks of code for the condition which checks if the player's effective position is an 'InvincibilityPotion' in lines 26 to 53 and 102 to 129 respectively, are repeated and both carry out the same actions.
 - Additionally the if conditions within the blocks, if not satisfied, sets `offset = getPosition()` in the else statements, which are repeated.
 - Unnecessary since `offset` is already initialised to `getPosition()`, and if the code get to this point, `offset` has not changed value and thus does not need to be set again.
- Additionally in the `ZombieToast` and `Mercenary` `move()` methods in lines 55 to 61 and 93 to 101, the code in `ZombieToast` where the player's effective potion is not an 'InvincibilityPotion' (given by the else statement) and the code in `Mercenary` which checks if the player's effective potion is an 'InvisibilityPotion' is the same.
 - `Mercenary` does also carry out `map.moveTo(this, nextPos)` after setting `nextPos` in lines 97 and 100, but this is unnecessary since it carries this action out after we leave the block of if statements.
 - Code executes random movement.
- In the `Enemy` and `ZombieToastSpawner` `onDestroy()` and `onMovedAway()` methods, they carry out the same actions.
- In line 63 of `ZombieToast`, `game.getMap` is called again despite already being stored as a variable in line 25.
- In line 90, 102 of `Mercenary`, `game.getPlayer` is repeatedly used despite being stored as a variable in line 84
- In lines 86 and 132 of `Mercenary`, `map.dijkstraPathFind` is called to set to `nextPos`.

ii. What Design Pattern could be used to improve the quality of the code and avoid repetition? Justify your choice by relating the scenario to the key characteristics of your chosen Design Pattern.

<Answer>

- Strategy pattern could be used.
 - By implementing a `MoveStrategy` Interface, the repeated functionality of the `InvincibilityPotion` check alongside the random move case within `ZombieToast` and `Mercenary`'s `move()` method can be extracted into respective classes implementing `MoveStrategy`
 - `InvincibilityPotion` check → `MoveFromPlayer`
 - `Mercenary` `InvisibilityPotion` check & `ZombieToast` random move → `MoveRandom`
 - `Mercenary` `map.dijkstraPathFind` → `MoveShortestPath`
 - `Mercenary` is adjacent to player and allied → `MovePlayerPreviousPos`

iii. Using your chosen Design Pattern, refactor the code to remove the repetition.

<Briefly explain what you did>


- Created an interface 'MoveStrategy' with the method 'getNextPos(GameMap map, Enemy enemy)' which returns a Position.
- Created strategy classes 'MoveShortestPath', 'MoveFromPlayer', 'MoveRandom' and 'MovePlayerPreviousPos' which implement MoveStrategy.
 - Each strategy class implementation of 'getNextPos' has its own distinct functionality that returns a Position.
- Adjusted the code accordingly in Mercenary and ZombieToast.
 - Initialised a MoveStrategy variable which would be set to the respective strategy depending on the required conditions.
 - Removed code carrying out actions that would determine nextPos and replaced with strategy setting.
 - Set nextPos = MoveStrategy.getNextPos(map, this) at end of conditions, eliminating repetition of setting nextPos for each condition.

b) Observer Pattern

Identify **one place** where the **Observer Pattern** is present in the codebase, and outline how the implementation relates to the **key characteristics** of the Observer Pattern.

- In Switch, the class stores a list of Bombs in which every Bomb is notified by the 'subscribe' or 'onOverlap' methods that are called by Switch.
 - The updating of the Bombs would occur when 'Switch' is activated.
 - Switch would thus be the subject whereas Bomb is an observer.
- This implementation relates to the key characteristics of the Observer pattern because by definition, the Observer pattern is 'used to implement distributed event handling systems in event driven programming'.
 - In this specific scenario, we could say that the event would be the Switch object being activated.
- Because Switch also maintains a list of Bombs which are notified by Switch when its 'state' becomes activated, occurring when it overlaps with a Boulder object, then this would also satisfy the Observer pattern where 'an object maintains a list of its dependents and notifies them automatically of any state changes in the subject, usually by calling one of their methods'.

c) Inheritance Design

 Links to your merge requests

https://nw-syd-gitlab.cseunsw.tech/COMP2511/23T2/teams/H11A_ABBA/assignment-ii/-/merge_requests/4

i. Name the code smell present in the above code. Identify all subclasses of **Entity** which have similar code smells that point towards the same root cause.

- The code smell present is of
 - duplicated code
 - the empty method only containing return is repeated in many subclasses
 - Refused Request
 - many of the functions hold no functionality for many of the subclasses of entity
 - *Feature Envy*
 - this issue is resultant from additional turn behaviours after moving an entity being executed in GameMap rather than inside the entity
 - Noted behaviours of three functions and affected subclasses below

	Empty Behaviours	Unique Behaviours	Same Behaviours (not inherited)
onDestroy	All except	None	<ul style="list-style-type: none"> Enemy & ZombieToastSpawner <pre> 1 @Override 2 public void onDestroy(GameMap map) { 3 Game g = map.getGame(); 4 g.unsubscribe(getId()); 5 } </pre>
onOverlap	<ul style="list-style-type: none"> exit wall buildable zombieTo astSpawner 	<ul style="list-style-type: none"> Boulder door player portal switch enemy → mercenary 	<ul style="list-style-type: none"> collectables → Arrow, key, sword, treasure, wood, potion(s) <pre> 1 if (entity instanceof Player) { 2 if (!((Player) entity).pickUp(this)) 3 return; 4 map.destroyEntity(this); 5 } </pre> <ul style="list-style-type: none"> also contained in bomb <ul style="list-style-type: none"> <pre> 1 if (state != State.SPAWNED) 2 return; 3 if (entity instanceof Player) { 4 if (!((Player) entity).pickUp(this)) 5 return; 6 subs.stream().forEach(s -> s.unsubscribe()); 7 map.destroyEntity(this); 8 } 9 this.state = State.INVENTORY; </pre>
onMovedAway	<ul style="list-style-type: none"> boulder door exit player portal wall buildable collectables <ul style="list-style-type: none"> arrow, bomb, key, sword, treasure, wood, potion enemy zombieTo astSpawner 	<ul style="list-style-type: none"> switch 	


- ii. Redesign the inheritance structure to solve the problem, in doing so remove the smells.

First the primary code smell came from the repeatedly overwritten methods of `onDestroy`, `onOverlap` and `onMovedAway` in `Entity` subclasses. To solve this issue, we changed its initial declaration as abstract methods in `Entity` to simply return to set a default behaviour for all classes

We then left the overrides of these methods which had unique implementations while deleting useless/copied default return behaviours. For instances where the same code was repeated such as in the "collectable" entities within the folder, we used the pull up method for the lines of code (for `onOverlap`) as stated above and created a new subclass of `Entity` called `Collectables` which contained this behaviour. We then made these classes extend `Collectables`.

However, it has been noted that further work must be done with `GameMap` to address this issue past its poor inheritance design.

d) More Code Smells

 Links to your merge requests

https://nw-syd-gitlab.cseunsw.tech/COMP2511/23T2/teams/H11A_ABBA/assignment-ii/-/merge_requests/6

Collectable entities are a big problem. We tried to change the way picking up items is handled, to be done at the player level instead of within the entity itself but found that we had to start making changes in heaps of different places for it to work, so we abandoned it.

- i. What design smell is present in the above description?

- This is an example of the design smell of shotgun surgery, where one change requires many different changes across code and classes.
- Shotgun surgery violates the open-closed principle, where classes should be 'closed for modification but open for extension' since the attempt to extend the code above required numerous classes/code to be modified.

- ii. Refactor the code to resolve the smell and underlying problem causing it.

- The Underlying problem was that picking up players relied on the originally repeated `onOverlap` method within collectable entities however this was resolved indirectly through the subclass refactoring done to the inheritance design with the creation of the `Collectables` class which pulled this up.
- Additionally,
 - the `canMoveOnto` method was pulled up into the `Collectables` class to minimise shotgun surgery since they all had the same behaviour
 - additionally, this behaviour in `onOverlap` is always true
 - ```
1 if (!((Player) entity).pickUp(this))
2 return;
```
    - so the conditional was removed, and the
      - ```
1 (Player) entity).pickUp(this);
2 map.destroyEntity(this);
```
 - Was condensed into a helper function to minimise repeated code in `Bomb`
 - all `onOverlay` behaviour was also delegated from `Player` to their respective `Entity` classes, with duplicate code handling mercenary overlay removed (since this was already implemented in the mercenary/enemy classes)

e) Open-Closed Goals

Links to your merge requests

https://nw-syd-gitlab.cseunsw.tech/COMP2511/23T2/teams/H11A_ABBA/assignment-ii/-/merge_requests/2

https://nw-syd-gitlab.cseunsw.tech/COMP2511/23T2/teams/H11A_ABBA/assignment-ii/-/merge_requests/8

i. Do you think the design is of good quality here? Do you think it complies with the open-closed principle? Do you think the design should be changed?

<Answer>

- I believe that the design is bad quality here because it is beginning to smell of complex switch operators.
 - In Goal, because it keeps track of 'types', there is lots of complexity regarding switches as well as the need to implement different constructors that take in different arguments.
 - Each of its methods has to constantly check the type of the Goal, in which it then carries out different actions.
 - Because constructors take in different arguments, for each different type, some variables stored within Goal are not used, e.g. only a treasure goal utilises 'target' while only an AND or OR goal only utilises 'goal1' and 'goal2'.
- The design does not comply with the open-closed principle. By definition, the open-closed principle states that 'classes should be closed for modification but open for extension'.
 - Goal clearly violates this principle since hypothetically speaking, if a new Goal type were to be introduced, then Goal would have to be modified (adding an additional case in the switch statements) in order to accommodate for this.
 - If this new type also had to keep track of different variables, then a new Constructor (and possibly new variables) would have to be implemented into the Goal class.
- Yes, I believe that the design should be changed.

ii. If you think the design is sufficient as it is, justify your decision. If you think the answer is no, pick a suitable Design Pattern that would improve the quality of the code and refactor the code accordingly.

<Briefly explain what you did>

- Composite pattern would improve the quality of the code.
 - Leaf nodes ExitGoal, BouldersGoal, TreasureGoal.
 - Compound nodes AndGoalNode, OrGoalNode.
- Create an interface 'GoalNode' which contains the methods 'achieved(Game game): boolean' and 'toString(Game game): String'.
- Create new classes 'ExitGoal', 'BouldersGoal', 'TreasureGoal', 'AndGoalNode', and 'OrGoalNode' which all implement the GoalNode interface.
 - Implement the necessary methods and store the necessary variables for each class, each carrying out their own distinct actions.
- Update Goal so that it takes in a GoalNode as the sole argument for its constructor, and remove the switch statements for the 'achieved' and 'toString' method. Replace with GoalNode.achieved(Game game) and GoalNode.toString(Game game) respectively.
- Update GoalFactory's method to take in the correct arguments when creating and returning a new Goal instance.
 - Create a private static method createGoalNode in GoalFactory which takes in same arguments as createGoal and returns a GoalNode depending on the goal type.
 - Used in createNode to return a new Goal with the respective GoalNode as an argument.

f) Open Refactoring

Merge request 1

https://nw-syd-gitlab.cseunsw.tech/COMP2511/23T2/teams/H11A_ABBA/assignment-ii/-/merge_requests/3

<Briefly explain what you did>


- As mentioned, the effects of potions (invisibility and invincibility) have been implemented using a State pattern but not effectively, therefore becoming poor design.
- 'PlayerState' stores two boolean variables (isInvincible and isInvisible), and utilises them in the methods isInvincible() and isInvisible().
 - Only used in Player.applyBuff(BattleStatistics) method.
 - Unneeded since we can determine this by looking at the 'inEffective' Potion variable stored within Player.
 - Additionally, PlayerState doesn't need to take them in as arguments for its constructor.
 - Change constructors, removing requirement for two boolean arguments.
 - Remove methods and variables from PlayerState.
 - Instead, have a method in PlayerState that returns BattleStatistics and encapsulates the same functionality as applyBuff for each individual PlayerState rather than having the logic in the block of if conditions.
 - Only BaseState would return origin.
- There is repetition of code and functionality in the transitionBase, transitionInvisible and transitionInvincible methods. This is because each PlayerState can transition to every other state given the correct potion to do so.
 - Unnecessary - remove all transition methods in subclasses and provide default transition methods in the PlayerState class rather than abstract ones.
- In Player, state transition methods are called in the 'triggerNext' method in Player, with each state changing according to the 'inEffective' Potion variable.
 - It would be better design to encapsulate this behaviour in a state transition method within PlayerState instead for improved readability and less convolution - easier to understand the conditions that would execute a state transition.
 - Remove the three transition methods in subclasses of PlayerState and implement default method 'transition(Potion): void' in PlayerState which checks the class of the potion, and changes the player state accordingly.
 - Update code in Player triggerNext() method to utilise this method instead.

 Merge request 2

https://nw-syd-gitlab.cseunsw.tech/COMP2511/23T2/teams/H11A_ABBA/assignment-ii/-/merge_requests/5

<Briefly explain what you did>

- In Potion, it implements the 'BattleItem' interface and subsequently, the 'applyBuff', 'use' and 'getDurability' methods.
 - When looking at where such methods are used, we can see that 'applyBuff' is used in the battle method in BattleFacade, specifically line 40. However, there is also a check beforehand, where if the BattleItem is a Potion, continue. Hence, Potion.applyBuff is never actually used anywhere.
 - 'use' doesn't actually have any functionality, just returning after it is called.
 - 'getDurability' is not used anywhere, hence it also is unnecessary.
- According to the specification, an InvincibilityPotion would just make the Player win the battle instantly, while an InvisibilityPotion prevents the Player from entering a battle in the first place.
- Hence, Potion does not require the implementation of the 'BattleItem' interface; it is redundant.
- Removed implementation of BattleItem interface from Potion.
 - Removed respective implemented methods applyBuff, use and getDurability from Potion and subclasses.
- Removed the check for BattleItem instance of Potion in BattleFacade battle() method

 Add all other changes you made in the same format

 Merge request 3

<Briefly explain what you did>

- As explained earlier when editing `onOverlap` and `onMovingAway` methods, the functions that call them in `GameMap` have extensive repeated code
 - the `moveTo` method direction and position variations have similar lines

```
1 // check canMoveTo using position
2 if (!canMoveTo(entity, position)) // OR if (!canMoveTo(entity, Position.translateBy(entity.getPosition(), direction)))
3     return;
4
5 // trigger events + move
6 triggerMovingAwayEvent(entity);
7 removeNode(entity);
8 entity.setPosition(position); entity.translate(direction);
9 addEntity(entity);
10 triggerOverlapEvent(entity);
```

- To remove repeated code, `public void moveTo(Entity entity, Direction direction)` was made to store a translated position as it does during the `canMoveTo` check and calls the `moveTo` method that takes in a position
- In the trigger events `triggerMovingAwayEvent` and `triggerOverlapEvent`, the code was identical, minus the functionality of adding either `onOverlap` or `onMovedAway` tasks on all appropriate entities

```
1 List<Runnable> callbacks = new ArrayList<>();
2 getEntities(entity.getPosition()).forEach(e -> {
3     if (e != entity)
4         callbacks.add(() -> e.onMovedAway(this, entity)); // e.onOverlap(this, entity)
5 });
6 callbacks.forEach(callback -> {
7     callback.run();
8 });
```

- To reduce repeated code, a helper function `triggerEvent(Entity entity, Consumer<Entity> eventHandler)` was created
 - since the repeated code differed only by the method that was used as a callback (i.e. `e.onMovedAway(this, entity)`) vs `e.onOverlap(this, entity)`, the `Consumer` interface was used to substitute the original lambda function
 - hence the line was replaced with `callbacks.add(() -> eventHandler.accept(e));`
 - the event handler was then defined using another lambda function with took in the `e` from `eventHandler.accept(e)`, copying the format of the original code

```
1 private void triggerMovingAwayEvent(Entity entity) {
2     triggerEvent(entity, (e) -> e.onMovedAway(this, entity));
3 }
```

Merge request 4

<Briefly explain what you did>

There were multiple violations of the Law of Demeter within `BattleFacade`, notably the calling of

1. `player.getBattleStatistics().getHealth();` and `enemy.getBattleStatistics().getHealth();`
 - a. this was solved by implementing a `getHealth()` method in both `Player` and `Enemy`
2. `for (BattleItem item : player.getInventory().getEntities(BattleItem.class)) {`
 - a. To solve this, method `getAllInventoryType(Class<T> clz)` was implemented in `Player` which replicated the above code

3. This was similarly present in `List<Mercenary> mercs = game.getMap().getEntities(Mercenary.class);`

a. To simplify the following helper method was added to Game

```
1 public <T extends Entity> List<T> getAllEntityType(Class<T> type) {  
2     return map.getEntities(type);  
3 }
```

c. to remove the temp field as well, the following was implemented

```
1 for (Mercenary merc : game.getAllEntityType(Mercenary.class)) {  
2     if (merc.isAllied()) {  
3         playerBuff = BattleStatistics.applyBuff(playerBuff, merc.getBattleStatistics());  
4     }  
5 }
```

4. `player.getBattleStatistics().setHealth(playerBattleStatistics.getHealth());` and

`enemy.getBattleStatistics().setHealth(enemyBattleStatistics.getHealth());`

a. the violation within `player/enemy.getBattleStatistics().setHealth` was resolved by similar to code smell 1, implementing a `setHealth()` method in both Player and Enemy

Merge request 5

https://nw-syd-gitlab.cseunsw.tech/COMP2511/23T2/teams/H11A_ABBA/assignment-ii/-/merge_requests/10

<Briefly explain what you did>

- The methods `getDurability()` and `use()` were copied between Shield and Bow so this was pulled up into buildables
- Additionally the durability field was shared so this was pulled up as well
- To solve the LOD violation `use()` → which was now in Buildables,

```
1 @Override  
2 public void use(Game game) {  
3     durability--;  
4     if (durability <= 0) {  
5         game.getPlayer().remove(this);  
6     }  
7 }
```

◦ a helper function was made in Game

```
1 public void removeFromPlayer(InventoryItem item) {  
2     player.remove(item);  
3 }
```

◦ and the LOD violation was subsequently replaced with the helper function

Merge request 6

https://nw-syd-gitlab.cseunsw.tech/COMP2511/23T2/teams/H11A_ABBA/assignment-ii/-/merge_requests/13

<Briefly explain what you did>

- Replaced the deprecated method call in Bomb's `onPutDown` method from
 - `translate(Position.calculatePositionBetween(getPosition(), p));`
 - to
 - `setPosition(p);`

- as stated in documentation
- throughout Bomb, MoveRandom, EntityFactory and Portal, there was an LOD Violation whereby
 - `.getPosition().getCardinallyAdjacentPositions()` was called despite position being stored in its parent class
 - in turn, this helper method was implemented in Entity (the parent class) as

```
1 public List<Position> getCardinallyAdjacentPositions() {
2     return position.getCardinallyAdjacentPositions();
3 }
```

- so it could be directly called by its subclasses

Merge request 7

https://nw-syd-gitlab.cseunsw.tech/COMP2511/23T2/teams/H11A_ABBA/assignment-ii/-/merge_requests/15

<Briefly explain what you did>

- noticed use and `getDurability` were same in Buildables and Sword class
 - in turn extracted these methods into newly created helper class `BattleItemBehaviour`
- then instantiated `BattleItemBehaviour` and delegated behaviours into this class

Merge request 8

https://nw-syd-gitlab.cseunsw.tech/COMP2511/23T2/teams/H11A_ABBA/assignment-ii/-/merge_requests/21

<Briefly explain what you did>

- battle items and player state buffs have a 'significant amount' of hard coding in their `applyBuff` methods.
 - resolve this by storing an a public final static `BattleStatistics` instance in Bow, since no variables determine the buff it gives.
 - store two public final static doubles in Shield for its attack magnifier and damage reducer stats respectively.
 - utilise the pre-existing public static variables in Sword for default stats.
 - store a public final static `BattleStatistics` instance in `InvisibleState` and `InvincibleState`, representing their respective buffs.
- `playerStates` `applyBuff` methods is not inherited from anywhere, though `battleItems` do implement the method.
 - create interface 'Buffing' which contains the public method `BattleStatistics applyBuff(BattleStatistics origin)`.
 - remove method from `BattleItem`, and instead make it extend the `Buffing` interface.
 - make `PlayerState` implement `Buffing`.

Merge request 9

https://nw-syd-gitlab.cseunsw.tech/COMP2511/23T2/teams/H11A_ABBA/assignment-ii/-/merge_requests/16

<Briefly explain what you did>

- noticed in `EntityFactory` (line 39), `MoveFromPlayer`, `MoveShortestPath` used
 - `map.getPlayer().getPosition()`
 - To solve this created `getPlayerPosition()` in `GameMap` as follows

```
1 public Position getPlayerPosition() {
2     return player.getPosition();
3 }
```

- also noticed in `Portal` (line 56), it used `pair.getPosition().getAdjacentPositions()`

- to solve created `getAdjacentPositions()` in Entity as follows

```
1 public List<Position> getAdjacentPositions() {
2     return position.getAdjacentPositions();
3 }
```

- also noticed in `MovePlayerPreviousPos` (line 9) it used `map.getPlayer().getPreviousDistinctPosition();`

- to solve created `getPlayerPreviousDistinctPosition()` in GameMap as follows

```
1 public Position getPlayerPreviousDistinctPosition() {
2     return player.getPreviousDistinctPosition();
3 }
```

Merge request 10

https://nw-syd-gitlab.cseunsw.tech/COMP2511/23T2/teams/H11A_ABBA/assignment-ii/-/merge_requests/18/

<Briefly explain what you did>

- Previously, our implementation of the MoveStrategy classes we implemented to solve the code smell of repeated code in 1.a.ii) did not solve the issues of LOD violations within the original code.
- As such, our original method was as follows
 - `public Position getNextPos(GameMap map, Enemy enemy)`
 - Requiring an initial `GameMap map = game.getMap();` within Mercenary, Spider, ZombieToast in order to pass on the parameter to a MoveStrategy
- To solve this issue, the interface and implementations were changed to `public Position getNextPos(Game game, Enemy enemy)`
 - and subsequent helper methods were ported into Game from GameMap such as since they both contained a player instance
 - `map.getPlayerPosition()` → `game.getPlayerPosition()`
 - `map.canMoveTo(enemy, moveX)` → `game.canMoveTo(enemy, moveX)`
 - `map.dijkstraPathFind(enemy.getPosition(), map.getPlayerPosition(), enemy);` → `game.dijkstraPathFind(enemy.getPosition(), game.getPlayerPosition(), enemy);`
 - This made it such that operations originally performed on the GameMap map instance were done through Game game, hence implementing linking methods
- Internally in Enemy, there was also an instance of an LOD violations in

```
1 @Override
2 public void onOverlap(GameMap map, Entity entity) {
3     if (entity instanceof Player) {
4         Player player = (Player) entity;
5         map.getGame().battle(player, this);
6     }
7
8     // this method also present in ZombieToastSpawner
9     @Override
10    public void onDestroy(GameMap map) {
11        Game g = map.getGame();
12        g.unsubscribe(getId());
13    }
```

- whereby operations were done on game and Game despite GameMap map being the only input parameter
 - to solve this issue, the following linking methods were created in map
 - `map.battlePlayer(this);`

- `map.unsubscribeFromGame(getId());`
- and hence replaced
- In Spider, the LOD violations in move
 - `List<Entity> entities = game.getMap().getEntities(nextPos);`
 - was solved by creating a linker method inside Game `game.getMapEntities(nextPos)`
 - `game.getMap().moveTo(this, nextPos);`
 - was solved by creating a linker method inside Game `game.moveTo(this, nextPos);`
 - both of which called the respective Map instance's method inside Game
- In ZombieToast and Mercenary, the LOD violations in
 - `map.getPlayer().getEffectivePotion()` were solved by creating linker helper method `game.getPlayerEffectivePotion()` inside Game
- In ZombieToastSpawner, the LOD violations in
 - `game.getEntityFactory().spawnZombie(game, this);`
 - were also solved by creating `game.spawnZombie(this);` in Game
 - which called upon the entityFactory instance's method
 - ```
1 public void interact(Player player, Game game) {
2 BattleItem weapon = player.getInventory().getWeapon();
3 weapon.use(game);
4 if (weapon instanceof Sword) {
5 game.getMap().destroyEntity(this);
6 }
7 }
```

    - were resolved by creating helper method `player.useWeapon(game);` in Player
      - this relied on another helper method made in Inventory
        - ```
1 public void useWeapon(Game game) {
2     BattleItem weapon = getWeapon();
3     weapon.use(game);
4 }
```

 - which was utilised in Player
 - The linker help method `game.destroyEntity(this)` was also created in Game

Merge request 11

https://nw-syd-gitlab.cseunsw.tech/COMP2511/23T2/teams/H11A_ABBA/assignment-ii/-/merge_requests/19

<Briefly explain what you did>

- In Door's private method 'hasKey', it has feature envy since it is more interested in Player's data rather than it's own (player inventory) and also violates the Law of Demeter since it gets a key from player inventory, then gets the key's number.
- Move the method and its logic into the Player class as a public method so that it can be utilised in other classes, taking in Door instead as argument rather than Player.
- Add getNumber method to Door to pass as an argument for player.hasKey method.
- Also violates law of demeter since we are getting the key, and then getting the key number.
 - add public method 'findFirstItem(itemType)' in Player which returns `inventory.getFirst(itemType)`.
 - use this method in hasKey method, taking in Key.class as argument.

Merge request 12

https://nw-syd-gitlab.cseunsw.tech/COMP2511/23T2/teams/H11A_ABBA/assignment-ii/-/merge_requests/20

<Briefly explain what you did>

- In BouldersGoal, EnemyGoal and ExitGoal's achieved() methods, game.getMap.getEntities is a violation of the Law of Demeter.
- ExitGoal achieved() method also gets player, and then gets player position, which is also a violation of Law of Demeter.
- Add method in Game which returns map entities depending on a class.
- Add method in Game which returns player position.
- Replace violations with the respective methods created.

Merge request 13

https://nw-syd-gitlab.cseunsw.tech/COMP2511/23T2/teams/H11A_ABBA/assignment-ii/-/merge_requests/22

<Briefly explain what you did>

- In GameMap, initRegisterSpawners spawning of spiders violates law of demeter (game.getEntityFactory().spawnSpider(game)).
- Add spawnSpider() method in Game and use this method in GameMap initRegisterSpawners method instead.

Merge request 14

https://nw-syd-gitlab.cseunsw.tech/COMP2511/23T2/teams/H11A_ABBA/assignment-ii/-/merge_requests/23

<Briefly explain what you did>

- Complex switch/if statements in the Inventory checkBuildCriteria method.
- in Player build() method, parse in the entity name String instead of checking if it equals shield, and modify the checkBuildCriteria method in Inventory to take in the String rather than boolean.
- Modify switch statement in the checkBuildCriteria method to check the String rather than forceShield, and encapsulate the logic for removing and building items in separate private methods.

Task 2) Evolution of Requirements 🧐

a) Microevolution - Enemy Goal

Links to your merge requests

https://nw-syd-gitlab.cseunsw.tech/COMP2511/23T2/teams/H11A_ABBA/assignment-ii/-/merge_requests/14

https://nw-syd-gitlab.cseunsw.tech/COMP2511/23T2/teams/H11A_ABBA/assignment-ii/-/merge_requests/17

Assumptions

- goals only evaluated after the first tick
- if getting to exit is one of the conditions conjoined with this one, it must be completed last.
- this specific goal (destroying a certain number of enemies or more, and all spawners) cannot be unachieved
 - an enemy cannot be undestroyed; there are no mechanics for that in place.

- a spawner also cannot be undestroyed, and new spawners cannot appear; the number of spawners that will exist is determined by the number of spawners that exist in the map at launch.
- (from Approved Assumptions) whether allies destroyed by a player-placed bomb count towards the enemy goal is undefined.
- destroying a spawner will not count towards the enemy goal.
- if all the cardinally adjacent cells to the spawner are walls, then the spawner will not spawn any zombies, but the spawner will not be considered destroyed.
- this is a standalone goal - it isn't a combination of two separate goal nodes using 'and'.

Design

- while this goal contains two conditions in order for it to be considered 'achieved', these two conditions cannot exist as standalone goals, hence, make this goal a standalone implementation of 'GoalNode' rather than subclass of 'AndGoalNode'
 - we'll call it 'EnemyGoal'
- will take in an integer 'target' representing the enemies that must be destroyed in its constructor.
 - Store integer 'enemiesDestroyed' in Game which tracks the enemies that are destroyed by Player.
 - increments when battle() method results in enemy getting destroyed
 - Have getter method in Game that returns the enemiesDestroyed, which then can be utilised in DestroyedGoalNode.
- the 'achieved' method will return true if both conditions are returning true and false otherwise.
 - condition 1 will return true if enemiesDestroyed >= target and false otherwise.
 - by using Game.getMap().getEntities(ZombieToastSpawner.class).size() to store in the integer 'spawnersRemaining', condition 2 will return true if spawnersRemaining == 0 and false otherwise.
 - this check is made automatically by the interact method in DungeonManiaController.
- currently, nothing happens when the player interacts with a ZombieToastSpawner using a sword. Implement behaviour in ZombieToastSpawner interact() method that destroys it if the player uses a sword on it.
 - one existing test contradicts this behaviour - fix the expected output

Test list

<Test List>

- no spawners and 0 enemies required to be destroyed
 - goal instantly achieved
- basic scenarios
 - no spawners and 1 enemy required to be destroyed
 - 1 spider
 - 1 mercenary
 - 1 zombie
 - 1 spawner (with default spawn interval 0 and can be interacted with) and 0 enemies required to be destroyed
 - 1 spawner (with default spawn interval 0 and can be interacted with) and 3 enemies required to be destroyed
- 1 spawner, 1 enemy required to be destroyed, no enemies on map
 - destroying spawner should not be considered destroying an enemy
 - goal cannot be achieved
- combining goal with every other type of goal
 - and
 - if combined with exit, exit must be completed last
 - or

Other notes

<Any other notes>

- add method in Game that returns the Map entities to satisfy Law of Demeter.

Choice 1 (Sunstone & More Buildables)



Links to your merge requests

https://nw-syd-gitlab.cseunsw.tech/COMP2511/23T2/teams/H11A_ABBA/assignment-ii/-/merge_requests/24

Assumptions

<Any assumptions made>

- Sceptre description: "Can be crafted with (1 wood OR 2 arrows) + (1 key OR 1 treasure) + (1 sun stone)".
 - If player has both 1 wood and 2 arrows, and also 1 key and 1 treasure, priority is undefined.
 - Which material chosen does not matter in this situation.
- Midnight armor can be crafted if "there are no zombies currently in the dungeon".
 - This doesn't mean that a zombie cannot spawn in the dungeon after the armor is crafted.
 - If there is a zombie spawner but no zombies, midnight armor is still craftable.
- Midnight armor attack and defence both reduces the player's stats if negative.
- Sunstones cannot be used to bribe mercenaries or assassins, but counts towards treasure count.
- Sunstones are able to be used to open doors, retaining itself after use.

From approved assumptions:

- The behaviour of a sceptre after use is undefined
- When trying to open a door with both a key and a sunstone in the player's inventory, it is undefined which entity will be used.
- The behaviour when `mind_control_duration` is ≤ 0 is undefined.
- When a mercenary or assassin can be bribed and mind controlled at the same time, which action will be taken after the player interacts with them is undefined.
- Whether a sunstone is preferred over keys or treasure when building a buildable item if both are available is undefined.
- Whether midnight armour counts as a weapon when destroying zombie toast spawners is undefined.
- Trying to Mind control/bribe an ally that is already mind controlled/bribe is undefined.

Design

<Design>

- In DungeonManiaController.java: for the interact() method, if the player does not have enough gold and does not have a sceptre and attempts to bribe/mind-control a mercenary, throw `InvalidActionException`.
- DungeonManiaController build() method: If the player does not have sufficient items to craft the buildable, or unbuildable for `midnight_armour` because there are zombies currently in the dungeon.
- Make a new class 'Sunstone' which extends Collectable.
 - Create an interface 'DoorOpener' which is implemented by both Key and Sunstone.
 - Create an interface 'Treasurable' which is implemented by both Treasure and Sunstone.

- Add method in Player and Inventory 'hasDoorOpener' which is used in Door to determine if player can open it on overlap, replacing the original method of hasKey.
 - Modify any helper methods accordingly.
- Modify pickUp method in Player which instead checks if the item is an instance of Treasurable to increment the treasure count.
- Can be used interchangeably with treasure or keys when building entities.
 - Modify checkBuildShield method in Inventory to get the number of Sunstones in Inventory, and implement the behaviour where it can be used interchangeably.
- Does not get removed if used to build or to open a door.
 - Add additional checks accordingly.
- Add Sunstone as a case in EntityFactory and GraphNodeFactory and return an instance of one accordingly.
- Make a new class 'Sceptre' which extends Buildable.
 - Constructor will take in an int mindControlDuration and store it.
 - Add buildSceptre as a method in EntityFactory.
 - will have to retrieve mind_control_duration from the config.
 - add sceptre as a case in Inventory checkBuildCriteria method
 - add private method checkBuildSceptre in Inventory which returns factory.buildSceptre and removes the items.
 - Add private boolean method in Mercenary canBeControlled which returns true if the player has a sceptre, and add the condition to isInteractable method.
 - In the interact method, check canBeControlled before calling bribe().
 - Store int timeMindControlled which increments by 1 every time move() is called.
 - Add method isMindControlled which gets the first sceptre from player inventory and returns true if timeMindControlled <= mindControlDuration.
 - Otherwise, set timeMindControlled = 0 and allied = false.
- Make a new class 'MidnightArmour' which extends Buildable.
 - Add buildMidnightArmour as a method in EntityFactory.
 - will have to retrieve midnight_armour_attack and midnight_armour_defence from the config.
 - add midnightArmour as a case in Inventory checkBuildCriteria method
 - add private method checkBuildMidnightArmour in Inventory which returns factory.buildSceptre and removes the items.
 - parse doubles midnight_armour_attack and midnight_armour_defence in MidnightArmour constructor and store them, using them in the applyBuff method.
 - set durability = null, and modify BattleItemBehaviour to not remove from player if durability = null.
- Add if condition in Inventory getBuildables method for both sceptre and midnight armour.

Changes after review

<Design review/ Changes made>

- Utilise state pattern for mercenary, with two different states instead of storing int timeMindControlled in Mercenary with method isMindControlled().
 - hostile state (default state)
 - can transition to controlled or bribed state.
 - allied state
 - if mindControlDuration == null, cannot transition out
 - transition to hostile state if timeMindControlled >= mindControlDuration
- There is no way to check if there are zombies on the map in the inventory.getBuildables() nor player.getBuildables method.
 - In GameBuilder buildMap method, where map sets player, also make player set map.

- Player stores an instance of GameMap.
- DoorOpener interface unnecessary since Sun stone doesn't have any behaviour when used to open a door.
- Sceptre is not technically a BattleItem (doesn't have a durability, apply any buffs, etc.), so perhaps modify inheritance structures
 - BattleBuildable → extends Buildable implements BattleItem
 - Buildable → extends Entity implements InventoryItem
 - Shield, Bow, Midnight Armour → extends BattleBuildable
 - Sceptre → extends Buildable

Test list

<Test List>

SunstoneTest

- Picking up sunstone in dungeon.
- Picking up sunstone in dungeon with treasure goal of 1.
- Using sunstone to open door → doesn't get removed
- Using sunstone to bribe mercenary → doesn't work.
- Building shield → doesn't get removed
- Building sceptre with 1 wood, 1 key and 1 sunstone → gets removed
- Building sceptre with 1 wood and 2 sunstones → only 1 gets removed
- Building midnight armour → gets removed

SceptreTest

- Building sceptre combinations.
- Not enough materials to build sceptre → throw exception
- Player does not have enough treasure and does not have a sceptre and attempts to bribe/mind-control a mercenary → throw exception
- Mind controlling mercenary from any position.
- Mind controlling a mercenary that is already allied.
- Mind controlling mercenary with duration of 1 second.
- Mind controlling mercenary with duration of 3 seconds.


MidnightArmourTest

- Building midnight armour combinations
- Not enough materials to build midnight armour → throw exception
- Zombies on the map when attempting to build → throw exception
- Using midnight armour in battle
 - 0 defence, 0 attack
 - negative defence, 0 attack
 - negative attack, 0 defence
 - positive defence, 0 attack
 - positive attack, 0 defence
 - fighting 100 enemies with 9999 defence
 - lasts forever

Other notes

<Any other notes>

Choice 2 (Logic Switches)

 Links to your merge requests

https://nw-syd-gitlab.cseunsw.tech/COMP2511/23T2/teams/H11A_ABBA/assignment-ii/-/merge_requests/27

Assumptions

- Self made
 - for CO_AND Logic
- All switches will be created in inactive state as well as all the logical entities
- Wires cannot be collected, and thus the Player can walk onto them without removing it from the map.
- Switch doors cannot be open with a Sun Stone
- Inactive/activate light bulbs and switch doors do not contribute to the logic rules of other entities; they are not considered 'conductors'.
- Wires themselves, do not have logic rules since they are not considered logical entities.
- Activations can only occur from movement (i.e boulder onto switch activation)
- Circuit breaks are undefined and hence only activation/deactivation by switch implementation is required by client

From approved/given assumptions:

- While player movement is not assessed on Lightbulb, assumed every other entity is able to move onto LightBulb
- Bomb breaking circuit is undefined behaviour
- Switch Door behaves the same as Door for Spiders
 - cannot be open with a key
- All switches will be created in inactive state as well as all the logical entities
- Any scenario where the order in which activated components perform their action is undefined
 - For example, this case where a logical bomb might activate and destroy parts of a circuit before other logical components are able to activate
- All logical entities will be created with the field `logic` which will be one of `and`, `or`, `xor`, or `co_and`. Note that light bulbs and switch doors will always be created with a logic field. Regular doors will never be created with a logic field, nor will floor switches or wires.
- Bombs may be created with the field `logic`. If they have this field, they are expected to be able to interact with other logical entities. Bombs created without this field function as they do in the MVP and do not need to interact with other logical entities.
- Switches are not logical entities, but instead activate cardinally adjacent logical entities, also known as conductors when they are switched on. When placed next to another already activated entity, they behave akin to wires in a circuit if they are switched on.
- Light bulbs and switch doors do not act as wires in a circuit.

Design

- Make a new interface LogicalRule and create classes AND, OR, XOR, CO_AND that implements them
 - contains boolean returning method check() which determines whether a LogicalEntity should be activated or not
- Make a new abstract class LogicalEntity that extends Entity
 - contains instance of LogicalRule as a means of Strategy Pattern
 - When initialised, takes in position alongside String logic
 - utilised switch operator to assign corresponding logicalRule to instance

- contains helper functions that pass on information to LogicalEntity for class to calculate =
 - amount of cardinally connected conductors
 - amount of activated cardinally connected conductors
- make subclasses SwitchDoor, LogicalBomb, LightBulb extends LogicalEntity
- contains abstract method `logicalUpdate(GameMap map)`
 - general pattern for subclasses is
 - `if(logicalRule.check)` subsequent logic that updates logical entity when updated
- LogicalEntities
 - LogicalBomb
 - contains instance of Bomb
 - calls `bomb.explode(map)` when logical rule conditions are met
 - SwitchDoor & LightBulb
 - holds boolean open/on which is toggled depending on `logicalRuleCheck()`
 - for SwitchDoor, `canMoveOnto()` method returns open
 - both classes have method that pass on open/on status which are utilised in NameConverter to properly pass on correct entity name
- Also created new class Conductor extends Entity
 - stores fields boolean activated
 - alongside int tickActivated which keeps track for CO_AND usage
 - has methods for
 - returning activation status
 - activation (which stores current tick)
 - and deactivation
 - Switch was moved to extends Conductor and new class Wire was created
- Conductors
 - Switch
 - modified onOverlap method to
 - call Conductor's activate method such that it could store tick activated
 - call `turnOnConnectedWires`
 - created new recursive methods to turn on connected wires
 - `turnOnConnectedWires(GameMap map, Position position, Set<Position> visited)`
 - iterated through all cardinal adjacent positions calling method `turnOnAdjacentWire(GameMap map, Position position, Set<Position> visited)`
 - which checked if tile contained wire and called its `activate()` method
 - modified onMovedAway to similarly
 - recursively call connected wires' `deactivate()` method
 - Wire
 - contained field `Set<Switch>` which stored the switches actively powering the wire
 - `activate()` method added the Switch responsible for powering on wire to set
 - also performed check to turn on and store tick activated if not already activated
 - `deactivate()` method removed Switch responsible for powering on wire from set
 - would then perform check to see if activated switches set was empty, and turn off if it was
 - combined layer of activation checking allowed for CO_AND implementation, as original tick activated (if another switch in same circuit was activated) would remain the same
- Also modified EntityFactory to take in logic field

- for bomb,
 - if empty, would call default Bomb() constructor
 - if filled, would call LogicalBomb() constructor
- Also in GameMap implemented new method `triggerLogicalEvents()` which occurred at the end of every movement
 - Assumptions
 - based on assumption that activations can only occur from movement (i.e boulder onto switch activation)
 - alongside assumption that circuit breaks are undefined and hence only activation/deactivation by switch implementation is required by client
 - used similar callback style as shown in `triggerOverlap/MoveAway` method
- Input related
 - implemented json cases to switch constructors in `GraphNodeFactory`, `EntityFactory`
- Return related
 - to ensure correct `DungeonResponse`'s were returned, modified `NameConverter`
 - for `SwitchDoor`, mimicked door behaviour, of adding `_open` based on open status
 - for `LightBulb`, used similar on status to add on `_on` or `_off`
 - for `LogicalBomb`, set to return "bomb"

Changes after review

<Design review/ Changes made>

- Noticed that Logical Bombs are still spawned as "collectable" on map
 - Thus Logical bombs should still share the same behaviour as normal bombs outside of their logical rules and functionality, being able to be picked up and placed down while maintaining their Logical Properties
- Thus `LogicalBomb` was required to now
 - be an `InventoryItem`
- Logical bombs instances storing an instance of a normal bomb could be problematic considering that there are no methods in `LogicalBomb` that would allow one to call the methods of the `Bomb` instance it stores.
 - Looking at the code, `LogicalBombs` have no `onOverlap` method, meaning that there is no way to pick up the bomb, and hence, also put it back down.
- Changes
 - First made `LogicalBomb` implements `InventoryItem`
 - implemented `onOverlap` method which mimicked the `Bomb` method
 - similar behaviour
 - player picks up item
 - difference being since the map stored a `LogicalBomb`
 - `LogicalBomb` instance was removed from map
 - setting bomb state required creation and calling of method `setState(State.INVENTORY)`
 - also implemented `onPutDown` method which
 - set new position for `LogicalBomb`
 - set new position for `Bomb` instance (such that explode would function in new location)
 - set state of `Bomb` `setState(State.PLACED)`, to prevent another pickup from occurring
 - `map.triggerLogicalEvents`, to check if newly placed logical bomb's conditions were met
- on Player side
 - created method `use(LogicalBomb logicalBomb, GameMap map)` → such that `LogicalBomb`'s could be used by player
- on Game side

- added LogicalBomb as a type of entity that could be used by the player in method `Game tick(String itemUsedId)`

Test list

- must be able to walk over wire
- For all LogicalEntity's (Bomb, Door, Lightbulb)
 - ensure on/off behaviour works correctly
 - Bomb → destroys surrounding entities (radius based)
 - Door → when activated allows movement, when not cannot move through
 - Lightbulb → on when activated, off when not
 - check if recursive activation/deactivation works
 - use long wire and disable and turn on for each entity type
 - See if activation occurs based on logical rule for each (and moveaway deactivation as well)

	OR	XOR	AND	CO_AND
(1/1) cardinally adjacent switch activates	true	true	false	false
(1/1) cardinally adjacent wire activates	true	true	false	false
Same-time, (2/2) cardinally adjacent wire activates	true	false	true	true
Same-time, (3/3) cardinally adjacent wire activates	true	false	true	true
Same-time, (4/4) cardinally adjacent wire activates	true	false	true	true
diagonally adjacent wire/switch	false	false	false	false

- For XOR
 - case: surrounded by off wires
 - turn on one → activate
 - turn on another → deactivated
- For AND
 - case: surrounded by 2 off wires
 - turn on one → not active
 - turn on another → activated
 - turn off one → deactivated
- For CO_AND
 - case: surrounded by 2 off wires
 - turn on one → not active
 - turn on another → not active


iii. turn off one → not active

- for checking activation of respective classes
 - Bomb → only need to check activation (entity disappears)
 - SwitchDoor → need to check open/close status
 - LightBulb → need to check on/off status

Other notes

<Any other notes>

Choice 3 (Insert choice) (If you have a 3rd member)

 Links to your merge requests

Assumptions

<Any assumptions made>

Design

<Design>

Changes after review

<Design review/ Changes made>

Test list

<Test List>

Other notes

<Any other notes>

 If you did more tasks add them here too

Task 3) Investigation Task !?

 Merge request 1

https://nw-syd-gitlab.cseunsw.tech/COMP2511/23T2/teams/H11A_ABBA/assignment-ii/-/merge_requests/14/diffs (issue stated below addressed initially while completing task 2a)

https://nw-syd-gitlab.cseunsw.tech/COMP2511/23T2/teams/H11A_ABBA/assignment-ii/-/merge_requests/29 (fixed mistake - removed the check for instanceof sword in ZombieToastSpawner interact method)

<Briefly explain what you did>

- From task 2(a), we recognised that the current MVP implementation of the logic regarding the interaction of ZombieToastSpawners by the Player wielding a weapon, did not match the requirements of the specification.
- According to the specification, “the Player can destroy a zombie spawner if they have a weapon and are cardinally adjacent to the spawner”.
 - Looking at the code however, we could see that interacting with the spawner used the weapon, reducing its durability, but did not actually remove the spawner from the Map; i.e. it doesn't get destroyed.
- Update the ZombieToastSpawner interact() method which calls the Game.destroyEntity() method to actually remove it from the map, and hence 'destroy' it.


- Update MVP implementation unit test in ZombieToast 'toastDestruction' to assert that no zombieToastSpawners exist on the map after the player interacts with it.

Merge request 2

https://nw-syd-gitlab.cseunsw.tech/COMP2511/23T2/teams/H11A_ABBA/assignment-ii/-/merge_requests/30

<Briefly explain what you did>

- We implemented logical bombs as an extension of Bomb under the assumption that a logical bomb would behave the same as our implementation of LogicalEntity rather than Bomb itself - that the Player is able to walk over them, but not pick them up.
- However, we realised that Logical Bombs, being an extension of Bomb, should be able to also be picked up and dropped down; logical bombs should share the same behaviour as Bomb outside of its logical rules and functionality.
- First made LogicalBomb implement InventoryItem
- Implemented onOverlap method which mimicked the Bomb method
 - similar behaviour
 - player picks up item
 - difference being since the map stored a LogicalBomb
 - LogicalBomb instance was removed from map
 - setting bomb state required creation and calling of method `setState(State.INVENTORY)`
- Implemented onPutDown method which
 - set new position for LogicalBomb
 - set new position for Bomb instance (such that explode would function in new location)
 - set state of Bomb `setState(State.PLACED)`, to prevent another pickup from occurring
 - `map.triggerLogicalEvents`, to check if newly placed logical bomb's conditions were met
- On Player side
 - created method `use(LogicalBomb logicalBomb, GameMap map)` → such that LogicalBomb's could be used by player
- On Game side
 - added LogicalBomb as a type of entity that could be used by the player in method `Game tick(String itemUsedId)`

 Add all other changes you made in the same format

Merge request 3

https://nw-syd-gitlab.cseunsw.tech/COMP2511/23T2/teams/H11A_ABBA/assignment-ii/-/merge_requests/32

<Briefly explain what you did>

- It is stated in the specification that "the Player can carry only one key at a time". However, when analysing the mvp implementation of keys, as well as the test that is supposed to check whether this behaviour is working as expected, we can see that it hasn't been properly implemented.

```

@Test
@Tag("4-4")
@DisplayName("Test player cannot pickup two keys at the same time")
public void cannotPickupTwoKeys() {
    DungeonManiaController dmc;
    dmc = new DungeonManiaController();
    DungeonResponse res = dmc.newGame(dungeonName:"d_DoorsKeysTest_cannotPickupTwoKeys", configName:"c_DoorsKeysTest_cannotPickupTwoKeys");

    assertEquals(expected:2, TestUtils.getEntities(res, type:"key").size());

    // pick up key_1
    res = dmc.tick(Direction.RIGHT);
    assertEquals(expected:1, TestUtils.getInventory(res, type:"key").size());
    assertEquals(expected:1, TestUtils.getEntities(res, type:"key").size());

    // pick up key_2
    res = dmc.tick(Direction.RIGHT);
    assertEquals(expected:2, TestUtils.getInventory(res, type:"key").size());
    assertEquals(expected:0, TestUtils.getEntities(res, type:"key").size());
}

```

- Under expected behaviour, the test should fail on lines 84 and 85, as we are only expected 1 key to be in the Player inventory, and because the Player cannot pick up the second key, 1 key should also still exist as an entity in the Map.
- To fix this, override the onOverlap method inherited by Key from Collectable, and add additional check to see whether the Player already has a key in their inventory before adding the key to their inventory and removing it from the map.

```

@Override
public void onOverlap(GameMap map, Entity entity) {
    if (entity instanceof Player && ((Player) entity).findFirstItem(itemType:Key.class) == null) {
        pickUpDestroy(map, (Player) entity);
    }

    return;
}

```

```

// pick up key_2
res = dmc.tick(Direction.RIGHT);
assertEquals(expected:1, TestUtils.getInventory(res, type:"key").size());
assertEquals(expected:1, TestUtils.getEntities(res, type:"key").size());
}

```

Update the MVP test to adhere to this behaviour



Merge request 4

https://nw-syd-gitlab.cseunsw.tech/COMP2511/23T2/teams/H11A_ABBA/assignment-ii/-/merge_requests/33

<Briefly explain what you did>

- In the specification, it states that "goals are only evaluated after the first tick". However, there is no code that carries out this functionality; there is nothing to check whether the first tick has occurred yet.
- This is evident in a test I had made for EnemyGoal, where there were no spawners and no enemies at all from the beginning of the game.

```

public class EnemyGoalsTest {
    @Test
    @DisplayName("Test goal - no enemies required no spawners")
    public void noEnemies() {
        DungeonManiaController dmc;
        dmc = new DungeonManiaController();
        DungeonResponse response = dmc.newGame("d_enemyGoalsTest_noEnemies",
            "c_basicGoalsTest_exit");
        List<EntityResponse> entities = response.getEntities();
        assertEquals(1, TestUtils.countEntityOfType(entities, "player"));

        // Assert goal met
        assertEquals("", TestUtils.getGoals(response));
    }
}

```

- Clearly, this goes against the expected behaviour since no ticks have passed, yet the goal has been evaluated as achieved.
- To fix this issue, call Game.getTick() method in the Goal.achieved() method.
 - If return value == 0, return false.

```

/**
 * @return true if the goal has been achieved, false otherwise
 */
public boolean achieved(Game game) {
    if (game.getPlayer() == null || game.getTick() == 0)
        return false;

    return goalNode.achieved(game);
}

```

- Update every GoalNode's toString method to also check for this

```

public String toString(Game game) {
    if (this.achieved(game) && game.getTick() != 0)
        return "";

    return "(" + goal1.toString(game) + " AND " + goal2.toString(game) + ")";
}

```

```

public String toString(Game game) {
    if ([this.achieved(game) && game.getTick() != 0])
        return "";

    return "(" + goal1.toString(game) + " OR " + goal2.toString(game) + ")";
}

```

```

public String toString(Game game) {
    if (this.achieved(game) && game.getTick() != 0) {
        return "";
    }

    return ":enemies";
}

```



```

public String toString(Game game) {
    if (this.achieved(game) && game.getTick() != 0)
        return "";
    return ":boulders";
}

```

- If tick is 0, return the String name of the goal.
- Update my test to recognise goal not achieved in 0th tick, but is achieved in 1st tick.

```

public class EnemyGoalsTest {
    @Test
    @DisplayName("Test goal - no enemies required no spawners")
    public void noEnemies() {
        DungeonManiaController dmc;
        dmc = new DungeonManiaController();
        DungeonResponse response = dmc.newGame(dungeonName:"d_enemyGoalsTest_noEnemies",
            |         configName:"c_basicGoalsTest_exit");
        List<EntityResponse> entities = response.getEntities();
        assertEquals(expected:1, TestUtils.countEntityOfType(entities, type:"player"));

        // Goal evaluted on first tick - assert goal not met
        assertEquals(expected:"enemies", TestUtils.getGoals(response));

        // Goal evaluted on first tick - assert goal met
        response = dmc.tick(Direction.RIGHT);
        assertEquals(expected:"", TestUtils.getGoals(response));
    }
}

```