# Design Rationale

## Zombie Attacks

**Zombies should be able to bite. Give the Zombie a bite attack as well, with a 50% probability of**

**using this instead of their normal attack. The bite attack should have a lower chance of hitting**

**than the punch attack, but do more damage – experiment with combinations of hit probability and damage that make the game fun and challenging. (You can experiment with the bite probability too, if you like.)**

This will be implemented in the AttackAction class and the bite attack will be executed if weapon is set to a certain weapon (eg. teeth). If actor is a zombie object, then at the start of the execute method, rand.nextBoolean() will be executed to determine whether weapon will be set for a normal attack or a bite attack (50/50 chance). The probability of the attack missing will be determined by checking rand.nextDouble() <= X, meaning there will be a (X\*100)% chance of missing and X will be determined on what weapon is. If weapon is for a normal attack (ie. fists or a weapon), then X=0.5, meaning a miss probability of 50%. If weapon is for a bite attack (ie. teeth), then X=0.7, meaning a miss probability of 70%.

Creating a new class for the bite attack was not chosen to be the best design as it would be similar to the AttackAction class and therefore require duplicated code which goes against the “Don’t repeat yourself” principle.

A successful bite attack restores 5 health points to the Zombie

The health restore will be done in the AttackAction class after the bite attack has been executed in the execute method. As the actor object is already passed as a parameter in the execute method, the health restore can be done with actor.heal(5).

If there is a weapon at the Zombie’s location when its turn starts, the Zombie should pick it up.

This means that the Zombie will use that weapon instead of its intrinsic punch attack (e.g. it might

“slash” or “hit” depending on the weapon)

A new class called ScanvengeBehaviour will be created which implements Behaviour. It will contain a getAction method that inspects the items at the actor’s location and picks up items that are a weapon. A ScanvengeBehaviour object will be the first element in the behaviours array in the Zombie class so that it will be executed at the start of its turn.

Creating a new class has been decided as the best design as it will group attributes and methods that depends on each other, following the “Group elements that must depend on each other together inside an encapsulation boundary” principle.

Every turn, each Zombie should have a 10% chance of saying “Braaaaains” (or something similarly Zombie-like)

This will be implemented in the Zombie class at the start of the playTurn method as this event is specific to only Zombie objects. A rand.nextDouble() <= 0.1 condition will be checked and if successful, the zombie will say “Braaaaains”.

## Beating up the Zombies

**Any attack on a Zombie that causes damage has a chance to knock at least one of its limbs off (I**

**suggest 25% but feel free to experiment with the probabilities to make it more fun)**

Within the AttackAction class, if a Zombie is dealt damage, then a rand.nextDouble() <= 0.25 condition will be checked and if successful, will knock least one of its limbs off. For each severed limb, a ZombieLimb will be created and passed on to a newly created DropLimbAction, where it will then be executed and the ZombieLimb will be dropped on the ground.

The number of limbs will be determined by x=rand.nextDouble, if x >= 0.15 then one limb will be knocked off, else if 0.05 <= x < 0.15 it will be two limbs, 0.01 <= x < 0.05 for three limbs and x < 0.01 for four limbs.

**On creation, a Zombie has two arms and two legs. It cannot lose more than these.**

There will be two extra attributes in the Zombie class to keep track on the number of arms and legs a Zombie has – armCount, legCount. These attributes will be set to 2 by default to represent two arms and two legs. There will be a condition that these attributes can only be an integer between 0 and 2 inclusive.

These attributes are stored in the Zombie class to keep it easily assessible when retrieving them and to ensure validity, following the “Classes should be responsible for their own properties” principle.

**If a Zombie loses one arm, its probability of punching (rather than biting) is halved and it has a**

**50% chance of dropping any weapon it is holding. If it loses both arms, it definitely drops any**

**weapon it was holding.**

In the AttackAction class, the probability that a normal attack will be executed for a Zombie will be determined by checking the Zombie’s armCount. If armCount is 2, then it’ll have the normal probability of 50%. If armCount is 1, then the probability will be halved to 25%. Else if armCount is 0, then the probability will be 0%. The probability of a bite attack is the remaining percent to make 100%.

It is assumed that the weapon a Zombie is holding is the first Weapon in its inventory. If a Zombie loses an arm, then a rand.nextBoolean() will be checked and if successful then the Zombie will drop the first Weapon in its inventory. If a Zombie loses both arms, then it will drop the first Weapon in its inventory.

**If it loses one leg, its movement speed is halved – that is, it can only move every second turn,**

**although it can still perform other actions such as biting and punching (assuming it’s still got at**

**least one arm)**

This will be implemented in the Zombie class. Zombie will have a Boolean attribute movedLastTurn that indicates whether the Zombie moved last turn. If Zombie only has one leg, then movedLastTurn is checked. If the one-legged Zombie has moved last turn, then HuntBehaviour and WanderBehaviour will not be used in the playTurn method.

**If it loses both legs, it cannot move at all, although it can still bite and punch**

Like above, however movedLastTurn will not be checked. HuntBehaviour and WanderBehaviour will not be used in the playTurn method if the Zombie has no legs.

**Lost limbs drop to the ground, either at the Zombie’s location or at an adjacent location (whichever you feel is more fun and interesting)**

The lost limbs will be dropped at an adjacent location to give the Zombie the option to move and pick up the weapon or to chase/attack a human. If the limb was dropped at the Zombie’s location, then the Zombie’s next turn is already determined as the Zombie will pick it up and end its turn. Limbs dropped on the ground will have an attribute indicating weather it was a leg or an arm that fell off for use in the ZombieClub and ZombieMace classes later

**Cast-off Zombie limbs can be wielded as simple clubs – you decide on the amount of damage they**

**can do**

When the Zombie loses a limb, the limb will be created as a ZombieLimb which inherits WeaponItem.

## Crafting weapons

Instead of making one class that can be either a zombie mace or zombie club there are two sperate classes. This is to avoid excessive unnecessary code as the attack damage would have to be adjusted based on what item it is which may add a unneeded reliability on ZombieLimb to determine what type of weapon it is.

**If the player is holding a Zombie arm, they can craft it into a Zombie club, which does significantly**

**more damage**

If the player does the CraftWeaponAction on a ZombieLimb item that is an arm it will delete the ZombieLimb object and replace it with a ZombieClub object which inherits from Weapon item.

**If the player is holding a Zombie leg, they can craft it into a Zombie mace, which does even more damage**

If the player does the CraftWeaponAction on a ZombieLimb item that is a leg it will delete the ZombieLimb object and replace it with a ZombieMace object which inherits from Weapon item.

## Rising from the dead

**if you’re killed by a Zombie, you become a Zombie yourself. After a Human is killed, and its corpse should rise from the dead as a Zombie 5-10 turns later.**

When a Human is unconscious for 5 or more turn, they have a 50% chance of dying and becoming a zombie. This is done through rand.nextInt(2). If they result is 0, they remain unconscious. If it is 1 the Human is dead. When they die the human object should be removed and is replaced by a Zombie object. If there are 10 turns where the Human is unconscious, they are automatically pronounced dead regardless of the random int.

## Farmers and food

**You must create a new kind of Human: Farmers. A Farmer shares the same characteristics and abilities as a Human**

Farmers will inherit from the human class

As A farmer will share the same characteristics and abilities of humans, we extend the human class as to adhere to the "Don't Repeat Yourself" Principle otherwise we would need to copy over the code from the human class.

**when standing next to a patch of dirt, a Farmer has a 33% probability of sowing a crop on it**

Farmers will interact with the GroundInterface as to avoid as many dependencies as possible (Reduce dependencies as much as possible principle). Since the ground interface is linked to Ground which is linked to Ground then Crop then Food. Therefore, the Farmer class only needs to interact with the GroundInterface and if the Farmer is standing on a patch of dirt the Dirt object will be replaced with a Crop object. This is to be done 33% of the time using rand.nextDouble(). If this value is less than 0.33 the Farmer will interact with the Ground to make the Dirt into a Crop, The crop object will have an attribute called ripe which will initially be set to False meaning it cannot be taken as food.

**left alone, a crop will ripen in 20 turns**

The Crop has an attribute counter initialised to 20. This counter will be decremented every game tick. If it reaches less than 0 the Crop object will have its ripe attribute changed to True allowing it to be taken as food later.

**when standing on an unripe crop, a Farmer can fertilize it, decreasing the time left to ripen by 10 turns**

The Farmer interacts with the GroundInterface and if the Farmer is standing on a Crop the Crop counter will decrease the counter by 10.

By interacting with the ground interface instead of the crop directly the crop does not need to store information such as if there is a farmer standing on it. In doing so the need for a dependency between farmer and crop is removed hence the reduce dependencies as much as possible principle is used.

**when standing on or next to a ripe crop, a Farmer (or the player) can harvest it for food. If a Farmer harvests the food, it is dropped to the ground. If the player harvests the food, it is placed in the player’s inventory**

The Farmer will interact with the ground interface. If it is a Crop Object that has its ripe attribute set as True, the Object will be deleted, and a Food object will be created. This Food object will have a String to describe the type of food it is a health attribute which will determine how many health points it will return to the player or human upon consumption.

**Food can be eaten by the player, or by damaged humans, to recover some health points**

If the players Health attribute is bellow 100 they can use the PickUpItem class to consume the Food if they are over it. Upon consumption the respective player or human will have the health attribute of the Food object added to the player health. Since the maximum health of a Human/Player is 100 if the result of eating the Food returns a health value greater than 100 its health will default to 100.