Overall Class Responsibilities

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  
• A successful bite attack restores 5 health points to the Zombie.**

Overall Class Responsibilities:

• In the original functionality of the game, a Zombie only attempts to punch, with a 50% chance of hitting and missing. The punch is returned by getIntrinsicWeapon( ) method in Zombie class.

• We plan to implement bite by potentially modifying the getIntrinsicWeapon( ) to return a punch or bite (rather than only returning bite).

• The weapon (punch or bite) is executed in AttackAction class in the execute( ) method. We plan to modify the execute( ) method to set a probabilites of hit and miss depending on whether the weapon is a punch or bite. If the weapon used is a bite and is successful, we communicate with class Zombie via the heal method( ) so that we can restore 5 health points.

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

Overall Class Responsibilities:

• In the original functionality of the game, a Zombie could not check to see if there is any weapons on the location of where it is standing.

• Now, the zombie picks up an item via behaviour class “PickUpBehaviour” which implements the Behaviour interface. PickUpBehaviour works by communicating with GameMap class via the method locationOf( ), which allows us to determine where the zombie is standing. We then communicate with Item class via method getItems( ) to check if there is any items that can be picked up. We then check if it is a valid weapon item, and if it is, we add it to the inventory of the zombie.

**• Every turn, each Zombie should have a 10% chance of saying “Braaaaains”**

Overall Class Responsibilities:

We implement the change in the playTurn method within the zombie class. We communicate with the Display class via the println( ) method to print “Braaains”. We set a 10% probability of printing this out though.

Beating up the Zombies

**• On creation, a Zombie has two arms and two legs. It cannot lose more than these.  
• Any attack on a Zombie that causes** **damage has a chance to knock at least one of its limbs off.   
• Lost limbs drop to the ground, either at the Zombie’s location or at an adjacent location   
• Cast-off Zombie limbs can be wielded as simple clubs – you decide on the amount of damage they can do.**

Overall Class Responsibilities:

• When a Zombie is created, the constructor in Zombie creates 2 arms and 2 legs by communicating with the ‘ZombieArm’ class and ‘ZombieLeg’ class. These limbs are stored in a collection that are an attribut of zombie.

• When a zombie is attacked, the hurt( ) method is called to reduce the zombie hitpoints (this is the original functionality of the system). We simply override this method, and set 25% chance to knock a limb off. A limb is knocked off by removing a limb from the Zombie collection at random.

• A zombie limb is a weaponItem (beause the ZombieLimb inherits weaponItem). Once a zombie limb is knocked off, we may potentially communicate with the GameMap class to drop the weaponItem onto the ground via the addItem( ) method.

**•** Because a zombie limbs inherits from WeaponItem, a Cast-off Zombie limbs can be wielded as simple clubs can now be wielded as a simple club.

**• 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.  
• If it loses one leg, its movement speed is halved. If it loses both legs, it cannot move at all, although it can still bite and punch.**

Overall Class Responsibilities:

**•** To correlate the probability of punching and number of arms, we simply modify the getItrinsicWeapon( ) method in the zombie class (no new classes are created). Instead of having a 50% chance of returning a punch and bite, we now check how many arms a zombie has. If a zombie has 1 arm, a chance of punch is 25% and chance of bite is 75%. If a zombie has 0 arms, a chance of punch is 0 and a chance of bite is 100% (although it can still miss).

**•** To make a Zombie have a 50% chance of dropping any weapon if it loses an arm and a 100% chance of dropping any items if it loses both arms, we override the hurt( ) method (no new classes are created). Within the override, we check if during the current attack, there was an arm lost, and if there was, we check that is the current number of arms the zombie has. We then set the corrresponding probabilities and drop the weapons accordingly.

**•** Zombie class can now reduce their movement speed via allowing/disallows various behaviours given the zombies condition. We know that the original functionality of the system includes the following: the word Class communicates to Zombie via the playturn( ) method in every turn of the actor, and passes to Zombie the last action performed. By communicating with world and storing the current number of legs, we can implement this functionality seamlessly, as we know the last action of the actor.

Rising from the dead

**As everybody knows, 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.**

In the AttackAction class, we check if a human actor is killed. If yes, the AttackAction class enables to place new human corpses which are instances of ResurrectableCorpseItem, and this is done by creating a CorpseItem class which inherits PortableItem class. Once the CorpseItem is created, the world class communicates to CorpseItem via the tick( ) method (as per the default functionality of the system), and this enables us to create a new zombie when the corpse is old enough by placing a zombie on the map.

Farmers and food

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

Overall Class Responsibilities:

• In the original classes of the game, there is no farmer class so we create need to create one.

• In the original functionality of the game, there is no sowing function, so we create “SowingBehaviour” which implements the Behaviour interface. “SowingBehaviour” works by getting the nearby location of the farmer and have a 33% probability to sow on a dirt.

**• Left alone, a crop will ripen in 20 turns.**

Overall Class Responsibilities:

• In the original classes of the game, there is no crop class so we create need to create one.

• Now, the crop class will have a method to tick so it can count the current turn and change its display character once it ripens.

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

Overall Class Responsibilities:

• In the original functionality of the game, there is no fertilizing function, so we create “FertilizeBehaviour” which implements the Behaviour interface. “FertilizeBehaviour” works by getting the current location of the farmer and decrease the time left to ripen if the farmer is standing on an unripe crop.

**• 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.**

Overall Class Responsibilities:

• In the original functionality of the game, there is no harvest function, so we create “HarvestBehaviour” which implements the Behaviour interface. “HarvestBehaviour” works by getting the nearby location including the current location of the farmer and the player and take different actions depends on the actor.

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

Overall Class Responsibilities:

• In the original functionality of the game, there is no eating function, so we create “SowingBehaviour” which implements the Behaviour interface. “EatingBehaviour” works by getting the actor’s inventory list and check is there any food inside this list. Then, it checks the actor’s current health point to qualify that the actor can take the food to heal.

Crafting Weapons

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

• In the original functionality of the game, there is no crafting function, so we create “CraftingBehaviour” which implements the Behaviour interface. “CraftingBehaviour” works by getting the actor’s inventory list and check is there any zombie arm and zombie leg inside this list. It will ask player that if he wants to craft the weapon to a different one.