CS PROJECT 3: REPORT

Aanish Patel Sikora

1. CLASSES: Public Member Functions
2. Actor

Actor is an abstract base class from which all other game actors are derived.

1. Actors constructor takes an image ID and starting position and passes it to GraphObject. It also sets m\_alive =true;
2. DoSomething is a pure virtual function because all actors are able to do something and this is unique to each derived object
3. isAlive returns m\_alive and indicates if the actor is alive or not
4. getWorld returns a pointer to the studentworld all the actors are alive in
5. newGetID is a pure virtual function and is used to identify which object each actor is. All actors have a unique ID
6. setWorld is called when each actor is constructed and is set to the StudentWorld object
7. The Destructor must be virtual so it is virtual.
8. Player

Player is derived directly from actor and is the main user class.

1. Player’s constructor takes a starting position, studentworld pointer and a number of sprayers. It makes itself visible, initializes the pointer using setWorld, sets the number of active sprayers to 0, sets its ability to walk through walls as false, gives itself two sprayers and 3 lives.
2. doSomething checks if player should be dead, checks if its on a goodie, decreases its special ability time and gets user input if it should move.
3. newGetID returns the player’s ID
4. givemorekill allows the player to drop more sprayers
5. isKiller checks if the player is currently able to drop more than two sprayers
6. giveLessKill gives the player the default kill power
7. numBS checks if it is possible for the player to drop another sprayer
8. addBS increases the number of bug sprayers dropped in the world
9. kill will kill the player
10. removeBS removes a bug sprayer from the world
11. isGhost checks if the player an walk through walls
12. setGhost makes the player a ghost
13. GhostTime, KillTime, decTKill, decTGhost all handle the amount of time the player is a ghost, killer for.
14. canMove checks to see if the player’s proposed move is valid
15. PBrick

PBrick is derived directly from Actor.

1. It’s constructor gives it a starting location and imageiD
2. DBrick

DBrick is derived directly from Actor.

1. It’s constructor gives it a starting location and imageID
2. Zumi

Zumi is base class from complex, simple zumi.

1. It’s constructor takes an imageID, starting position, studentworld pointer and speed. It initializes the pointer, chooses a random direction to move and sets its time to move to the speed parameter. .
2. Move will check if this turn is valid turn to move(zumi can only move 1 turn out of x). It will then attempt to move in its current direction if it is valid. If not, it chooses new random direction and waits a full x number of ticks before moving.
3. setDir sets a new direction for the zumi to move
4. getDir returns the current direction of the zumi
5. getSpeed determines current waiting period for zumi
6. decWait decreases waiting time
7. Simple Zumi

Is derived from Zumi

1. Its constructor takes a starting position and pointer and speed. It makes itself visible.
2. doSomething checks to see if its alive and kills itself if there is bugspray. Otherwise it usings zumis move algorithm.
3. Complex Zumi

Is Derived from Zumi

1. its constructor takes a starting position and pointer and speed. It makes itself visible.
2. doSoemthing: it kills the player is touching it. It finds the player pos. if player pos within seach distance, initiate search algorithm.

Else move like normal zumi

1. pathExists and explore are the search algorithm. Path exists takes the player pos and the zumi pos. explore is not member function of complex zumi. The pathexists function iterates through the current map and creates a character array with ‘X’ representing obstacles and ‘.’ Denoting empty. It then creates a queue of pointers to a structure called Point. Each point instance has a pair of x,y coordinates and a pointer to a point. Pathexists will initialize the current zumi position and sets its pointer to null. It will then mark this position as visited and push it on the queue. Then, while the queue is not empty and the exit is not found, it will pop the first position, check if it’s the players position and if not, it will explore left, right, up and down. Explore will check if the position is valid and if so push’s it onto the queue, marks it as marked and set its pointer to the first point.

What essentially happens it that each point pushed onto the queue knows which point preceeded it. Therefore, at the end a while loop is used to traverse from the final point to the first point by the shortest path possible.

1. DoomedToDie

This class is derived from actor and it is the base class from every timed object (Bug sprayer, bug spray and all goodies).

1. Its constructor takes an imageID, starting position, pointer to studentworld and the time it has left alive.
2. doSomething and newGetID are pure virtual as each derived class will have its own implementation.
3. decTime reduces the time till death.
4. getTime returns time till death.
5. setTime sets the time to some parameter.
6. BugSprayer

Is derived from DoomedToDie

1. Its constructor takes a starting position, time to die and pointer to studentworld. It makes itself visible.
2. doSomething checks if object is alive or not, decreases time to die. It checks if it has been hit by bugspray. If so, it detonates itself causing a chain reaction. Otherwise, it checks if it should detonate, if so, it adds bug spray in a “’+”formation, two units in each direction. Bug spray is only added to valid positions. Not on permanent bricks and not on goodies. It also plays a sound. It then kills itself
3. BugSpray

Is derived from DoomedToDie.

1. Its constructor takes a starting position, time to die and pointers to studentworld and bugsprayer. It makes itself visible.
2. Dosomething checks if its alive or not, kills itself if time has run out. It then checks if there is another actor or player and if so it kills it.
3. GameExit

Is derived from Actor

1. It’s constructor takes a starting position. And it sets itself to invisible.
2. Goodie

Is derived from DoomedToDie.

1. It’s constructor takes a starting position, imageID, studentworld pointer and time to die.
2. doSomething checks if time has elapsed and if so it kills itself.
3. LifeGoodie

Is derived from Goodie

1. It sets its visibility to true
2. Ghost

Is derived from Goodie

1. It sets its visibility to true
2. Killer

Is derived from Goodie

1. It sets its visibility to true
2. StudentWorld
3. The constructor doeesn’t do anything except set the number of zumis currently alive to 0. This is because the init and cleanup functions take care of all the variables.
4. There are 10 functions that are trivial. All they do is return the level parameters loaded from the data file to make them available to other classes.
5. Init: Loads the correct level. If past level 0 and there is not next level, player has won. It then loads level parameters from data file. It then runs two for loops through the maze and if it encounters an actor, allocates memory and creates it and adds pointer to a vector.
6. Move: Lets the player dosomething and lets every alive actor to dosomething. Checks if player is alive or not, if not return so. Looks for dead actors and removes them from the game. It then updates the display text. It then checks if there are no more zumis, and checks if the player is on the exit, if so return so.
7. Cleanup: it deletes the player pointer, makes everything invisible. It then removes each one from the vector and sets the number of zumi to 0.
8. setDisplayText: uses stringstreams to display level, lives,bonus and score in specified format.
9. killActor: it checks if current actor is zumi and calls killZumi. Else it runs through the vector of actor pointers, finds the correct one and makes it invisible, deletes the memory and removes from vector. If it’s a bug sprayer, call players removeBS method.
10. KillZumi: runs through the loop and finds correct actor pointer from the vector. It increases score, decreases number of zumi, kills object and deletes the memory and removes from vector. Then it checks if it is valid to drop a goodie, and will randomly decide to drop either life,ghost or killer goodie. Then, if there are no more zumi, it shows the exit.
11. betterCheck: is a helper function that tells me what actor currently occupies a specified position.
12. addBugSpray: checks if it is possible to drop a bug spray and if so, it does. Doesn’t drop if currently at permanent brick, goodie or exit.
13. addKillerGoodie: adds a killer goodie randomly if it is valid to do so.
14. addLifeGoodie: adds a life goodie randomly if it is valid.
15. AddBugSprayer: adds bug sprayer
16. showExit: it finds the exit object in the vector and makes it visible
17. killPlayer: plays a sound and kills the player
18. destructor: deletes player pointer and every other dynamically allocated memory.
19. I finished everything. The only problem was related to the ticksperComplexZumi move parameter. For some reason, it this value is less than 3 when I open Level 0, the complex zumis do not move simply. They will move in the complex way when I get close to them but when I am outside their smell radius, they won’t move for ticksperComplexZumi< 3. I had no idea how to fix this so I just set the code such that the value has a minimum of 10.
20. No assumptions made that weren’t specified from the spec.
21. Testing was performed incrementally and on each function as it was added (SEVERAL TIMES).

I wrote stubs for every class and member function I knew I would need from the beginning. As I went deeper, I added helper functions. Testing involved messing with the level data files and cout and cerr and debug.

CLASS ACTOR: Not directly tested (IT IS ABC), tested by testing derived classes.

CLASS PLAYER: Tested by verifying correct user input, verifying ability to drop weapon, ability to die.

CLASS PBRICK AND DBRICK: Tested by ensuring no actor can move onto these bricks, tested destroyable brick by verifying that I could destroy it with bug spray.

CLASS ZUMI: Mostly tested simple move by verifying its movements and comparing to provided function. Complex Zumi testing was the most involved and hard. I used to cout to see what the pathExists algorithm was doing and to see if the correct path was being chosen.

CLASS GOODIE: Verified by setting each probability to 100 in turn and checking if life was gained, could walk through walls, could drop more sprayers for limited time.

CLASS EXIT: This was tested by verifying that it correctly showed up and let me win the game under correct circumstances.   
CLASS BUG SPRAYER: Tested by verifying that I could correctly drop it and it would release spray.

CLASS BUG SPRAY: Verified that it would kill actors sharing its square.

CLASS STUDENTWORLD: Tested by playing the game hundreds of times and comparing each game iteration with provided game’s gameplay.

on. ake care of all the vairabl other dynamically allocated memory. occupies a specified position. ake care of all the vairablon. ake care of all the vairabl other dynamically allocated memory. occupies a specified position. ake care of all the vairablon. ake care of all the vairabl other dynamically allocated memory. occupies a specified position. ake care of all the vairabl