BOĞAZİÇİ UNIVERSITY
INDUSTRIAL ENGINEERING

IE 201: Intermediate Programming

TERM PROJECT – "Plants vs Zombies Game"

Design Report

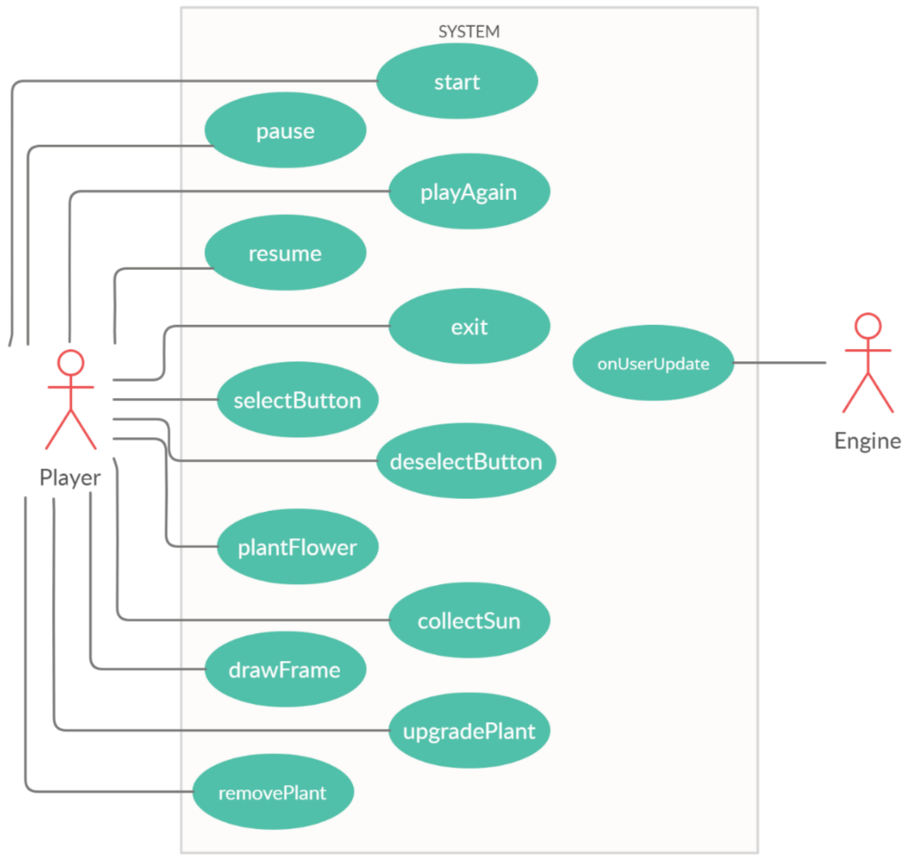
14.01.2021

Group 4

Baran Kırkgöz	2018402048
Barış Arslan	2019402129
Eyyüp Taşçı	2019401108

CONTENTS

USE CASE DIAGRAM

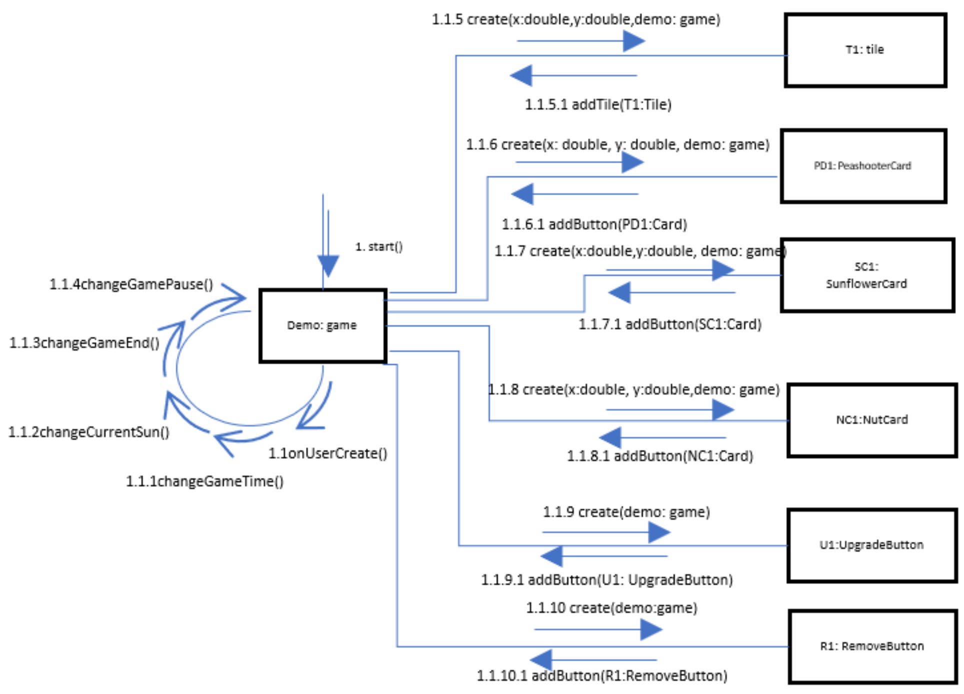


Use case diagram simply describes the expected behaviors of the system from the perspective of the users. As can be seen from the figure 1, our use case diagram consists of ten use cases, nine for human player and one for engine itself.

Firstly 'pause' is the case when the player wants to temporarily stop the game, and 'resume' is the case to continue playing. When the game is over the player can either restart the game or quit the game. These behaviors correspond to 'playAgain' and 'exit' cases, respectively. While playing the game; the player can select a button and deselect the selected card or collect the sun by clicking. 'selectCard', 'deselectCard', 'collectSun' use cases represent these behaviors, respectively again. Player can plant the selected card, upgrade or remove a plant by choosing buttons. 'plantFlower', 'upgradePlant', 'removPlant' use cases represent these behaviors, respectively. Player can move the cursor upon the graphics of the game and this behavior represented by 'drawFrame' use case. Lastly computer operates the game by running a loop, 'onUserUpdate' use case is the time when the computer runs this loop.

COLLABORATION DIAGRAMS

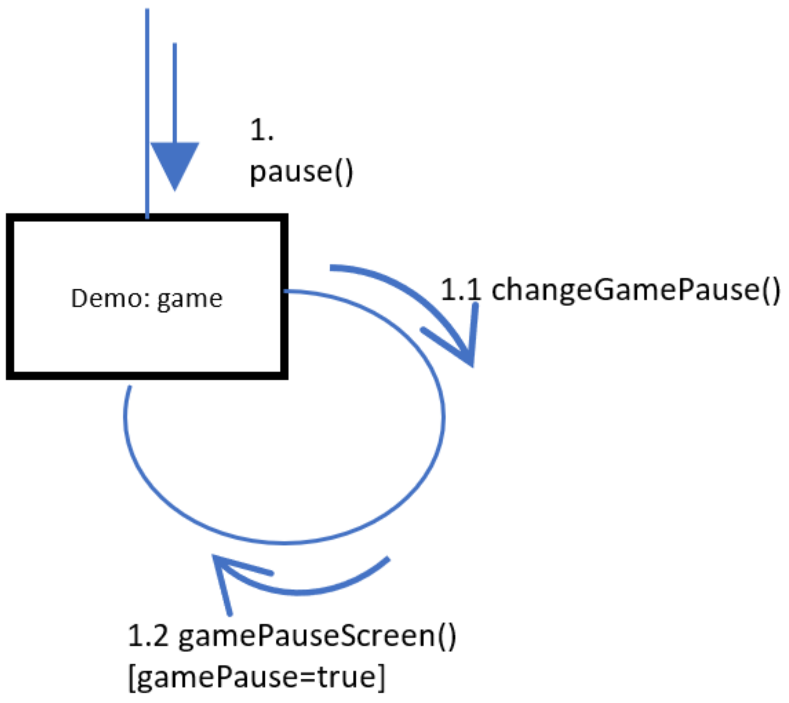
1. Start


1.1.5 create(x:double,y:double,demo: game)
T1: tile
1.1.5.1 addTile(T1:Tile)
1.1.6 create(x: double, y: double, demo: game)
PD1: PeashooterCard
1.1.6.1 addButton(PD1:Card)
1.1.7 create(x:double,y:double, demo: game)
SC1: SunflowerCard
1.1.7.1 addButton(SC1:Card)
1. start()
1.1.4changeGamePause()
Demo: game
1.1.3changeGameEnd()
1.1.2changeCurrentSun()
1.1onUserCreate()
1.1.1changeGameTime()
1.1.8 create(x:double, y:double,demo: game)
NC1:NutCard
1.1.8.1 addButton(NC1:Card)
1.1.9 create(demo: game)
U1:UpgradeButton
1.1.9.1 addButton(U1: UpgradeButton)
1.1.10 create(demo:game)
R1: RemoveButton
1.1.10.1 addButton(R1:RemoveButton)

Start is called. At the beginning of the game onUserCreate() is called. Then, gameTime is set to 0 by using changeGameTime, currentSun is set to 100 by using changeCurrentSun, gameEnd is set to false by using changeGameEnd(), gamePause is set to false by using changeGamePause. Then objects are created one by one. Firstly, game object sends create message to object then object sends a add message to game.

COLLABORATION DIAGRAMS

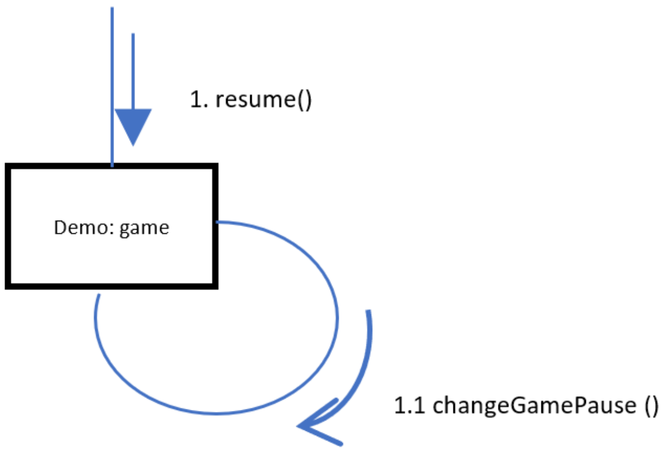
2. Pause



The player can pause the game by pressing the key P. After P is pressed, gamePause is set to true by changeGamePause. After gamePause become true, player sees the pause screen.

COLLABORATION DIAGRAMS

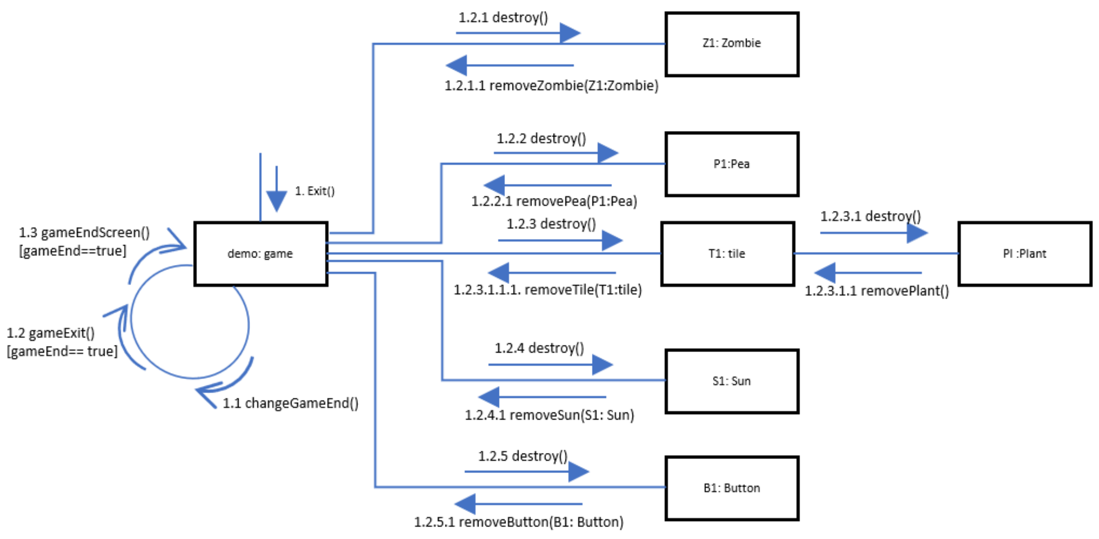
3. Resume



After game is paused, player can either continue or exit the game. When player choosed to resume the game, gamePause is set to false by changeGamePause and player continues to game.

COLLABORATION DIAGRAMS

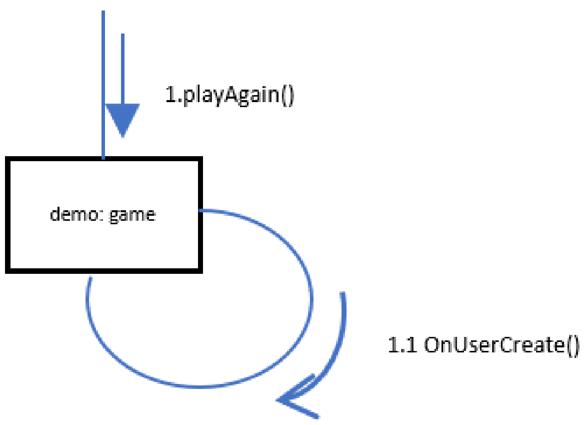
4. Exit


1.2.1 destroy()
Z1: Zombie
1.2.1.1 removeZombie(Z1:Zombie)
1.2.2 destroy()
P1:Pea
1. Exit()
1.2.2.1 removePea(P1:Pea)
1.2.3 destroy()
1.2.3.1 destroy()
1.3 gameEndScreen() [gameEnd==true]
demo: game
T1: tile
Pl :Plant
1.2.3.1.1.1. removeTile(T1:tile)
1.2.3.1.1 removePlant()
1.2 gameExit() [gameEnd== true]
1.2.4 destroy()
S1: Sun
1.1 changeGameEnd()
1.2.4.1 removeSun(S1: Sun)
1.2.5 destroy()
B1: Button
1.2.5.1 removeButton(B1: Button)

A player can exit the game in two ways. Player can press button E or after player lost the game, the game ends. After the game ends, gameEnd is set to true by changeGameEnd. Then gameEnd becomes true; game object sends message to objects by using destroy. And objects send message to game with remove function.

COLLABORATION DIAGRAMS

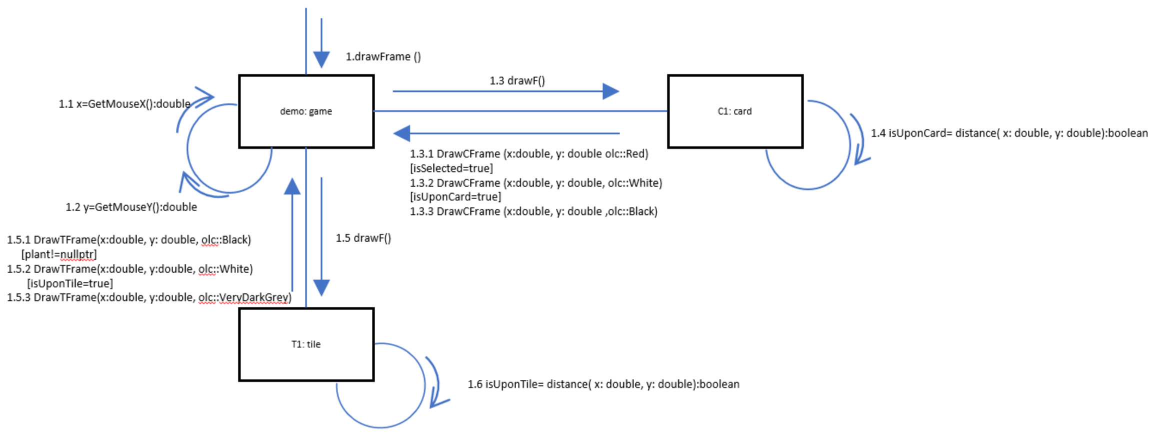
5. Play Again



After a player looses the game, they can choose to play the game again. After player chooses to play again, onUserCreate is called and gameTime, currentSun, gameEnd and gamePause are set to default and objects are created one by one.

COLLABORATION DIAGRAMS

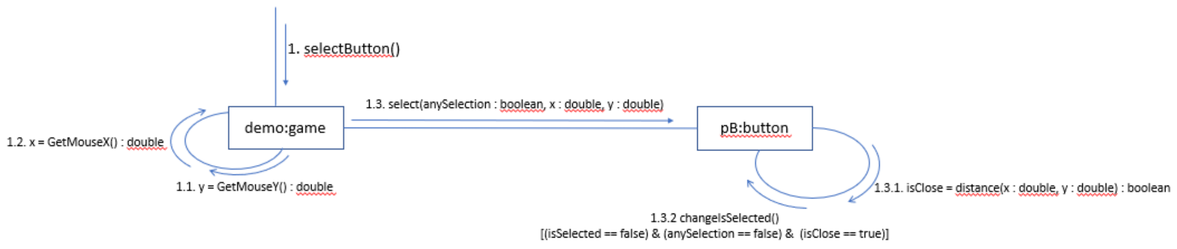
6. Draw Frame


1.drawFrame ()
1.3 drawF()
demo: game
C1: card
1.1 x=GetMouseX():double
1.4 isUponCard= distance(x: double, y: double):boolean
1.3.1 DrawCFrame (x:double, y: double olc::Red)
[isSelected=true]
1.3.2 DrawCFrame (x:double, y: double, olc::White)
[isUponCard=true]
1.3.3 DrawCFrame (x:double, y: double ,olc::Black)
1.2 y=GetMouseY():double
1.5.1 DrawTFrame(x:double, y: double, olc::Black)
[plant!=nullptr]
1.5.2 DrawTFrame(x:double, y:double, olc::White)
[isUponTile=true]
1.5.3 DrawTFrame(x:double, y:double, olc::VeryDarkGrey)
1.5 drawF()
T1: tile
1.6 isUponTile= distance(x: double, y: double):boolean

Frame can be drawed around tiles or cards. Around cards, if card is selected; frame is red, if mouse is on card but not selected, frame is white; else black. If there there is plant on a tile, frame of a tile becomes black; if mouse on tile, frame becomes white; otherwise the frame of a tile is very dark grey.

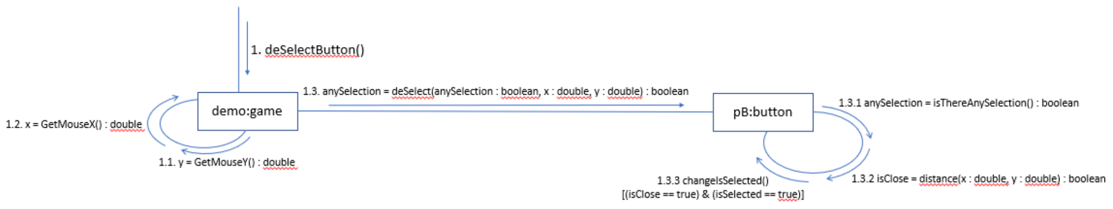
COLLABORATION DIAGRAMS

7. Select Button


1. selectButton()
1.3. select(anySelection : boolean, x : double, y : double)
demo:game
pB:button
1.2. x = GetMouseX() : double
1.1. y = GetMouseY() : double
1.3.1. isClose = distance(x : double, y : double) : boolean
1.3.2 changeIsSelected()
[(isSelected == false) & (anySelection == false) & (isClose == true)]

When select button is called, firstly location of mouse returned in game object. Then game object sends a select message to button object with anyselection, x and y inputs. Then the x and y of mouse are checked if they are upon button by using isClose. If the button is not selected, if there is no any selection and if the mouse is upon button, is selected becomes true by using changeIsSelected.

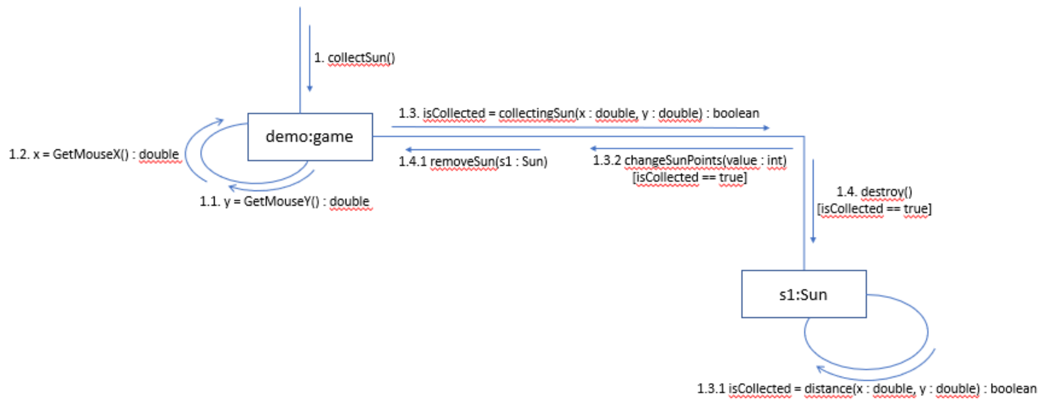
8. Deselect Button


1. deSelectButton()
1.3. anySelection = deSelect(anySelection : boolean, x : double, y : double) : boolean
demo:game
pB:button
1.2. x = GetMouseX() : double
1.1. y = GetMouseY() : double
1.3.1 anySelection = isThereAnySelection() : boolean
1.3.2 isClose = distance(x : double, y : double) : boolean
1.3.3 changeIsSelected()
[(isClose == true) & (isSelected == true)]

When deselect button is called, firstly location of mouse returned in game object. Then game object sends deselect message to button object with anyselection return type.Then, isThereAnySelection makes anySelection true if the button is selected, otherwise anySelection remains false. Then the x and y of mouse are checked if they are upon button by using isClose. If the mouse is upon button and button is selected, isSelected is set to false by using changeIsSelection.

COLLABORATION DIAGRAMS

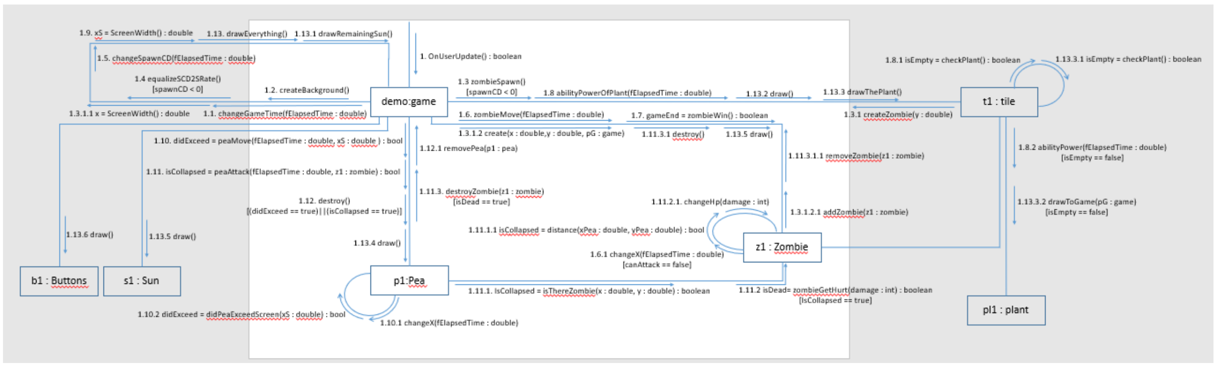
9. Collect Sun


1. collectSun()
demo:game
1.3. isCollected = collectingSun(x : double, y : double) : boolean
1.2. x = GetMouseX() : double
1.4.1 removeSun(s1 : Sun)
1.3.2 changeSunPoints(value : int) [isCollected == true]
1.1. y = GetMouseY() : double
1.4. destroy() [isCollected == true]
s1:Sun
1.3.1 isCollected = distance(x : double, y : double) : boolean

After collectSun is called, firstly location of mouse is returned in game object. Then the location of mouse is sent to sun object by using collectSun. Then the distance of mouse is checked. If the mouse is upon, value of the sun is sent to game and the current sun changes. After the suns are collected, sun is removed from game.

COLLABORATION DIAGRAMS

10.OnUserUpdate


1.9. xS = ScreenWidth() : double
1.13. drawEverything()
1.13.1 drawRemainingSun()
1. OnUserUpdate() : boolean
1.5. changeSpawnCD(fElapsedTime : double)
1.4 equalizeSCD2SRate()
[spawnCD < 0]
1.2. createBackground()
1.3 zombieSpawn()
[spawnCD < 0]
1.8 abilityPowerOfPlant(fElapsedTime : double)
1.13.2 draw()
1.13.3 drawThePlant()
1.8.1 isEmpty = checkPlant() : boolean
1.13.3.1 isEmpty = checkPlant() : boolean
demo:game
t1 : tile
1.3.1.1 x = ScreenWidth() : double
1.1. changeGameTime(fElapsedTime : double)
1.6. zombieMove(fElapsedTime : double)
1.7. gameEnd = zombieWin() : boolean
1.3.1 createZombie(y : double)
1.3.1.2 create(x : double,y : double, pG : game)
1.11.3.1 destroy()
1.13.5 draw()
1.10. didExceed = peaMove(fElapsedTime : double, xS : double) : bool
1.12.1 removePea(p1 : pea)
1.11.3.1.1 removeZombie(z1 : zombie)
1.8.2 abilityPower(fElapsedTime : double)
[isEmpty == false]
1.11. isCollapsed = peaAttack(fElapsedTime : double, z1 : zombie) : bool
1.11.3. destroyZombie(z1 : zombie)
[isDead == true]
1.12. destroy()
[(didExceed == true) | |(isCollapsed == true)]
1.11.2.1. changeHp(damage : int)
1.3.1.2.1 addZombie(z1 : zombie)
1.13.3.2 drawToGame(pG : game)
[isEmpty == false]
1.11.1.1 isCollapsed = distance(xPea : double, yPea : double) : bool
1.13.6 draw()
1.13.5 draw()
1.13.4 draw()
1.6.1 changeX(fElapsedTime : double)
[canAttack == false]
z1 : Zombie
b1 : Buttons
s1 : Sun
p1:Pea
1.11.1. IsCollapsed = isThereZombie(x : double, y : double) : boolean
1.11.2 isDead= zombieGetHurt(damage : int) : boolean
[IsCollapsed == true]
pl1 : plant
1.10.2 didExceed = didPeaExceedScreen(xS : double) : bool
1.10.1 changeX(fElapsedTime : double)

OnUserUpdate executes most of the program commands. It is called by the engine repeatedly, and it executes the timed actions of the game. Therefore, the first function it calls does change game time by adding the time passed in the last call. Then it draws the background.

To spawn zombies, firstly spawnCD variable is checked. If it is less than zero zombieSpawn function of the tile is called by the game to get the y coordinate of the tile. Tile gives its y coordinate as parameter in the createZombie function. Then game gets its ScreenWidth for x coordinate and it creates zombie in those x and y coordinates. After zombie is created, it adds itself to the zombie vector in the game. If spawnCD is less than zero, it is equalized to spawnRate. spawnCD is decreased by the fElapsedTime value even though zombie is not created.

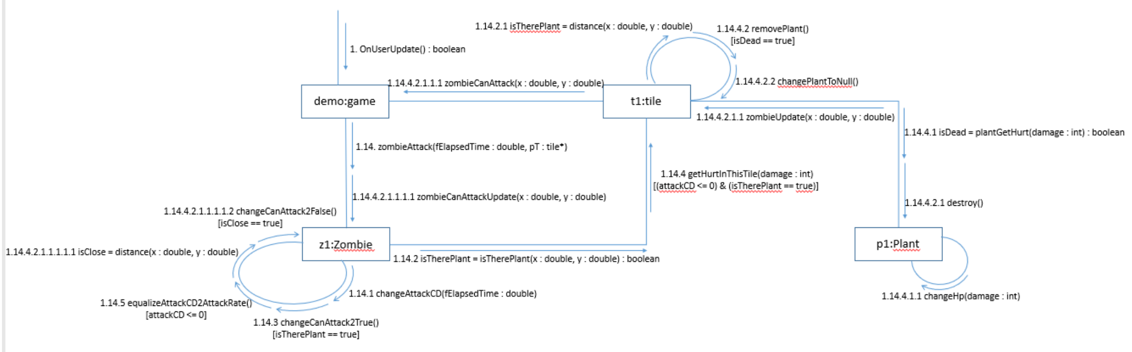
To move zombies, fElapsedTime variable is passed to zombie in the function zombieMove. If canAttack variable of zombie is false, zombie changes its x coordinate by the algebraic manipulations of the fElapsedTime value. After zombies moved, game checks that is any zombie passed the left end of screen by the function zombieWin. Zombie compares its x value and returns true if it is passed the line.

To move peas, fElapsedTime and ScreenWidth variables are passed to tile. Pea changes its x coordinate by the algebraic manipulations of the fElapsedTime value. After that Pea compares its x value with ScreenWidth and returns true if it is passed the line.

For the attack of the peas, fElapsedTime and Zombie variables are passed to Pea. Pea gives its coordinates to the Zombie. Zombie checks whether Zombie is hit by pea. If it did, then Zombie changes its hp. If the hp becomes less than zero, game destroys the zombie and zombie removes itself from the zombie vector in the game.

When pea is passed the line or reached any zombie, then game destroys the pea and removes itself from the pea vector. Lastly game draws every object.

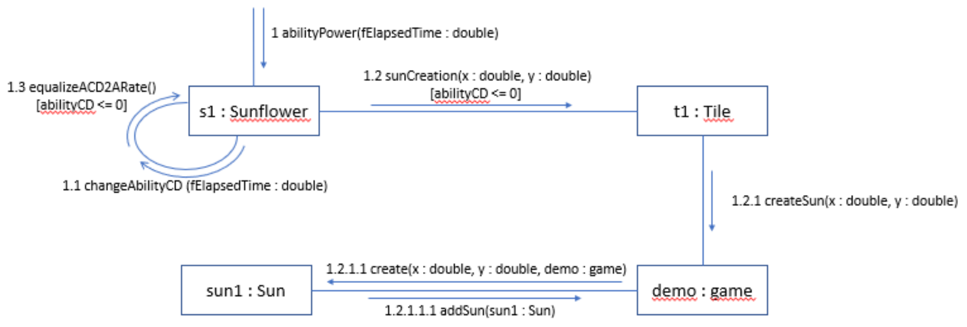
COLLABORATION DIAGRAMS

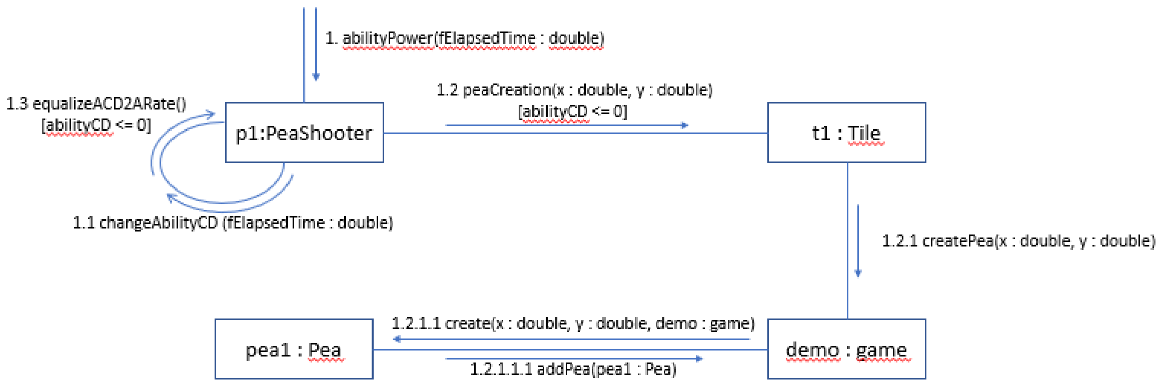

1. OnUserUpdate() : boolean
1.14.2.1 isTherePlant = distance(x : double, y : double)
1.14.4.2 removePlant()
[isDead == true]
1.14.4.2.2 changePlantToNull()
1.14.4.2.1.1.1 zombieCanAttack(x : double, y : double)
demo:game
t1:tile
1.14.4.2.1.1 zombieUpdate(x : double, y : double)
1.14.4.1 isDead = plantGetHurt(damage : int) : boolean
1.14. zombieAttack(fElapsedTime : double, pT : tile*)
1.14.4 getHurtInThisTile(damage : int)
[(attackCD <= 0) & (isTherePlant == true)]
1.14.4.2.1.1.1.1 zombieCanAttackUpdate(x : double, y : double)
1.14.4.2.1 destroy()
1.14.4.2.1.1.1.1.2 changeCanAttack2False()
[isClose == true]
z1:Zombie
p1:Plant
1.14.4.2.1.1.1.1.1 isClose = distance(x : double, y : double)
1.14.2 isTherePlant = isTherePlant(x : double, y : double) : boolean
1.14.1 changeAttackCD(fElapsedTime : double)
1.14.4.1.1 changeHp(damage : int)
1.14.5 equalizeAttackCD2AttackRate()
[attackCD <= 0]
1.14.3 changeCanAttack2True()
[isTherePlant == true]

For the attack of the zombies, fElapsedTime and tile variables are passed to Zombie. Firstly, attackCD is changed. Zombie gives its coordinates to the tile and tile checks whether there is any plant that Zombie can give damage. If there is, zombie changes its canAttack. If attackCd is less than zero and there is a plant to give damage, Zombie passes its damage to the plant through tile and plant changes its hp by the damage value. If the plant hp become less than zero, it is destroyed by the tile. If Plant dies, zombies canAttack are updated.

COLLABORATION DIAGRAMS

11.Ability Power

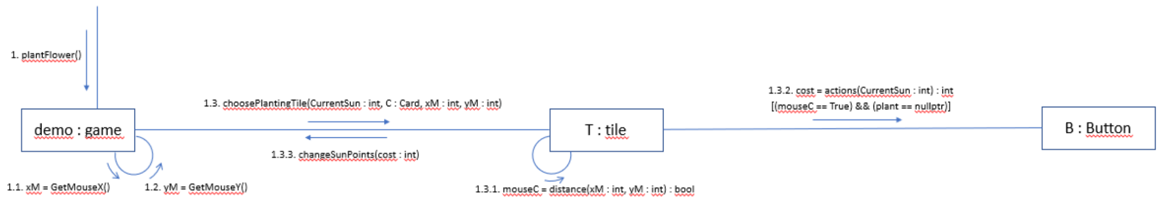

1 abilityPower(fElapsedTime : double)
1.3 equalizeACD2ARate()
[abilityCD <= 0]
s1 : Sunflower
1.2 sunCreation(x : double, y : double)
[abilityCD <= 0]
t1 : Tile
1.1 changeAbilityCD (fElapsedTime : double)
1.2.1 createSun(x : double, y : double)
sun1 : Sun
1.2.1.1 create(x : double, y : double, demo : game)
demo : game
1.2.1.1.1 addSun(sun1 : Sun)

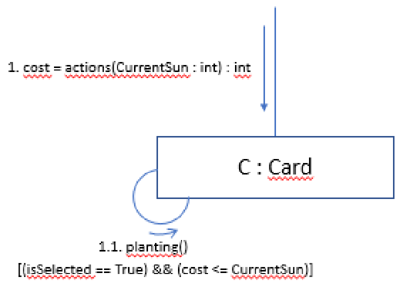

1. abilityPower(fElapsedTime : double)
1.3 equalizeACD2ARate()
[abilityCD <= 0]
p1:PeaShooter
1.2 peaCreation(x : double, y : double)
[abilityCD <= 0]
t1 : Tile
1.1 changeAbilityCD (fElapsedTime : double)
1.2.1 createPea(x : double, y : double)
pea1 : Pea
1.2.1.1 create(x : double, y : double, demo : game)
demo : game
1.2.1.1.1 addPea(pea1 : Pea)

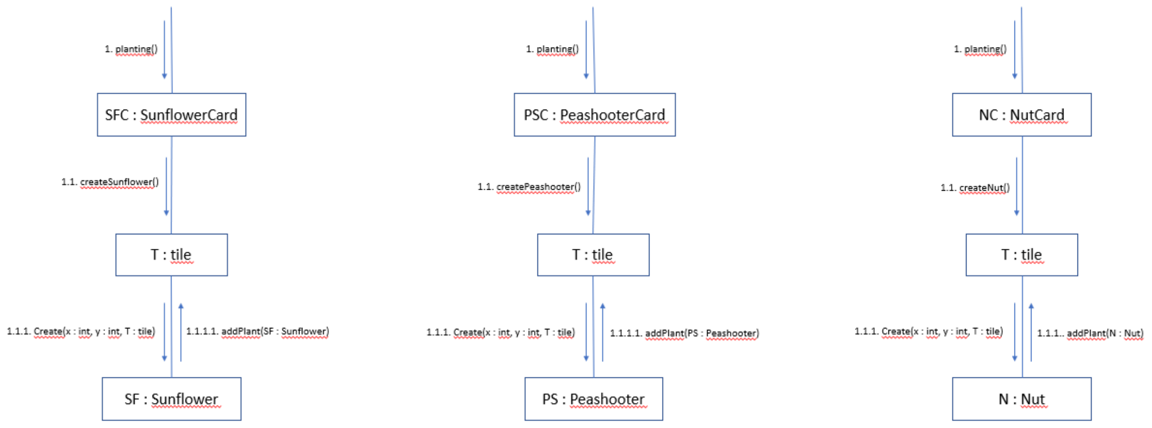
To trigger the abilities of the plants, firstly fElapsedTime variable is passed to tile in the function abilityPowerOfPlant. If there exists a plant in the tile, then fElapsedTime variable is passed to plant. There is a polymorphic structure in the function of abilityPower. Plant passes its x and y coordinates to game through tile and if the plant is Sunflower, game creates sun in those x and y coordinates. If the plant is Peashooter, pea is created instead of sun. AbilityCD is updated in the abilityPower function.

COLLABORATION DIAGRAMS

12.Plant Flower


1. plantFlower()
demo : game
1.1. xM = GetMouseX()
1.2. yM = GetMouseY()
1.3. choosePlantingTile(CurrentSun : int, C : Card, xM : int, yM : int)
1.3.3. changeSunPoints(cost : int)
T : tile
1.3.1. mouseC = distance(xM : int, yM : int) : bool
1.3.2. cost = actions(CurrentSun : int) : int
[(mouseC == True) && (plant == nullptr)]
B : Button


1. cost = actions(CurrentSun : int) : int
C : Card
1.1. planting()
[(isSelected == True) && (cost <= CurrentSun)]


1. planting()
SFC : SunflowerCard
1.1. createSunflower()
T : tile
1.1.1. Create(x : int, y : int, T : tile)
1.1.1.1. addPlant(SF : Sunflower)
SF : Sunflower
1. planting()
PSC : PeashooterCard
1.1. createPeashooter()
T : tile
1.1.1. Create(x : int, y : int, T : tile)
1.1.1.1. addPlant(PS : Peashooter)
PS : Peashooter
1. planting()
NC : NutCard
1.1. createNut()
T : tile
1.1.1. Create(x : int, y : int, T : tile)
1.1.1.. addPlant(N : Nut)
N : Nut

plantFlower use case allows the player to plant flower after selecting a plant card.

Player selects a tile to plant and “plantFlower” message is sent to the game. Game gets the mouse locations and sends a message with input parameters of mouse locations and current sun value to the tile.

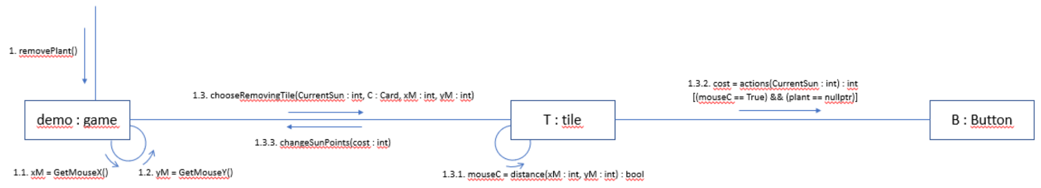
COLLABORATION DIAGRAMS

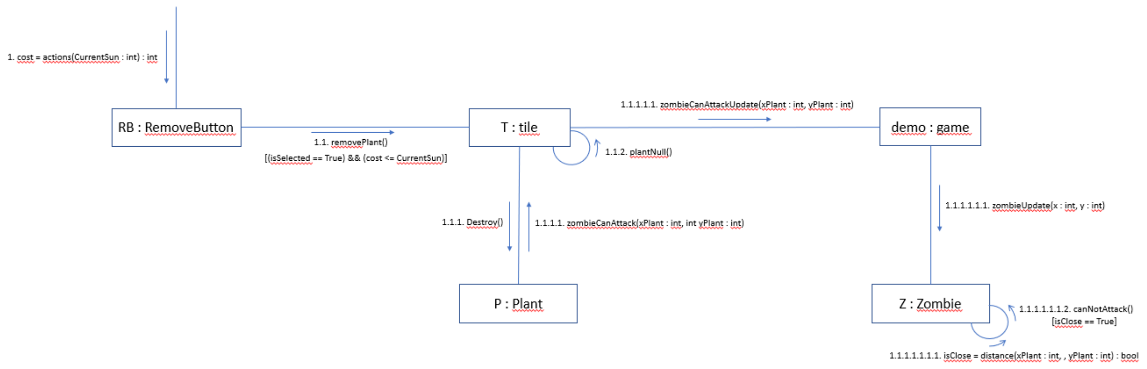
Tile compares the mouse locations with its coordinates. If the conditions are satisfied, it sends a message to the card to activate planting which returns the cost of the card. Then, tile sends the cost to the game to change the current sun.

If card is selected and current sun is sufficient, each type of the plant cards uses the same virtual “planting” function of the card and tells tile to create its own plant. Tile creates the plant.

COLLABORATION DIAGRAMS

13.Remove Plant


1. removePlant()
demo : game
1.1. xM = GetMouseX()
1.2. yM = GetMouseY()
1.3. chooseRemovingTile(CurrentSun : int, C : Card, xM : int, yM : int)
1.3.3. changeSunPoints(cost : int)
T : tile
1.3.1. mouseC = distance(xM : int, yM : int) : bool
1.3.2. cost = actions(CurrentSun : int) : int
[(mouseC == True) && (plant == nullptr)]
B : Button


1. cost = actions(CurrentSun : int) : int
RB : RemoveButton
1.1. removePlant()
[(isSelected == True) && (cost <= CurrentSun)]
T : tile
1.1.2. plantNull()
1.1.1. Destroy()
1.1.1.1. zombieCanAttack(xPlant : int, int yPlant : int)
P : Plant
1.1.1.1.1. zombieCanAttackUpdate(xPlant : int, yPlant : int)
demo : game
1.1.1.1.1.1. zombieUpdate(x : int, y : int)
Z : Zombie
1.1.1.1.1.1.2. canNotAttack()
[isClose == True]
1.1.1.1.1.1.1. isClose = distance(xPlant : int, , yPlant : int) : bool

removePlant use case allows the player to remove a plant after selecting the remove button.

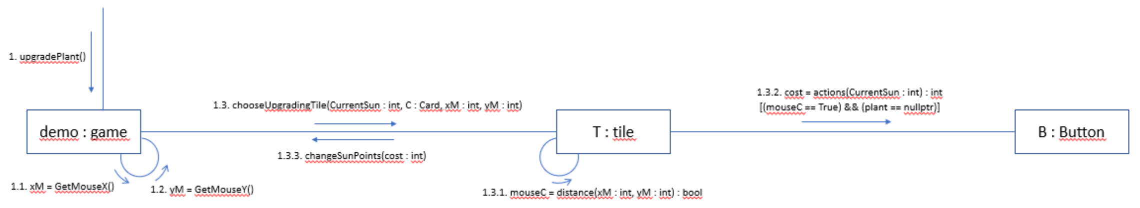
Player selects a plant to remove from the tile and “removePlant” message is sent to the game. Game gets the mouse locations and sends a message with input parameters of mouse locations and current sun value to the tile.

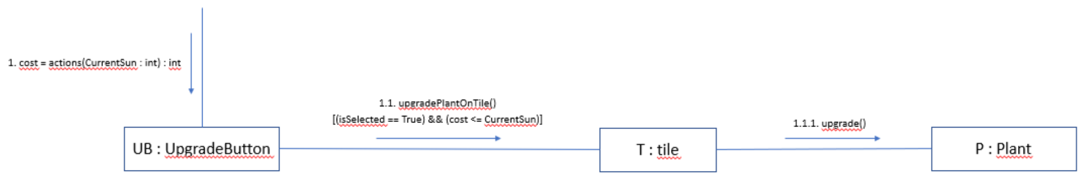
Tile compares the mouse locations with its coordinates. If the conditions are satisfied, it sends a message to the remove button to activate removing which returns the refund of the removal. Then, tile sends the refund to the game to change the current sun.

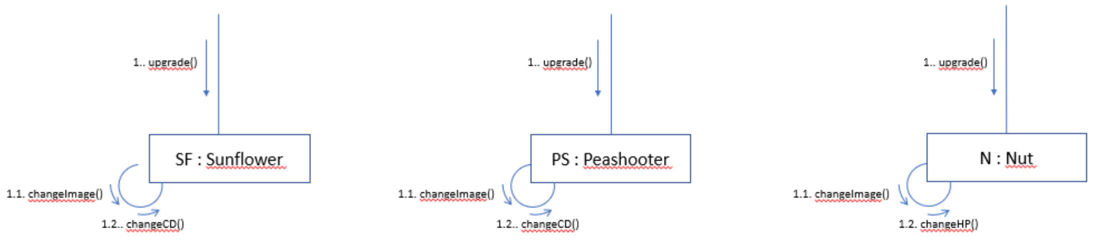
If the remove button is selected, it tells tile to remove its own plant. Tile destroys its plant and gets a message from the plant to inform zombies about the removal. Another message from the tile is sent to the game to update all zombies. Game calls “zombieUpdate” function and zombies turn their canAttack values to false if the removed plant is close to them.

COLLABORATION DIAGRAMS

14.Upgrade Plant


1. upgradePlant()
demo : game
1.1. xM = GetMouseX()
1.2. yM = GetMouseY()
1.3. chooseUpgradingTile(CurrentSun : int, C : Card, xM : int, yM : int)
1.3.3. changeSunPoints(cost : int)
T : tile
1.3.1. mouseC = distance(xM : int, yM : int) : bool
1.3.2. cost = actions(CurrentSun : int) : int
[(mouseC == True) && (plant == nullptr)]
B : Button


1. cost = actions(CurrentSun : int) : int
UB : UpgradeButton
1.1. upgradePlantOnTile()
[(isSelected == True) && (cost <= CurrentSun)]
T : tile
1.1.1. upgrade()
P : Plant



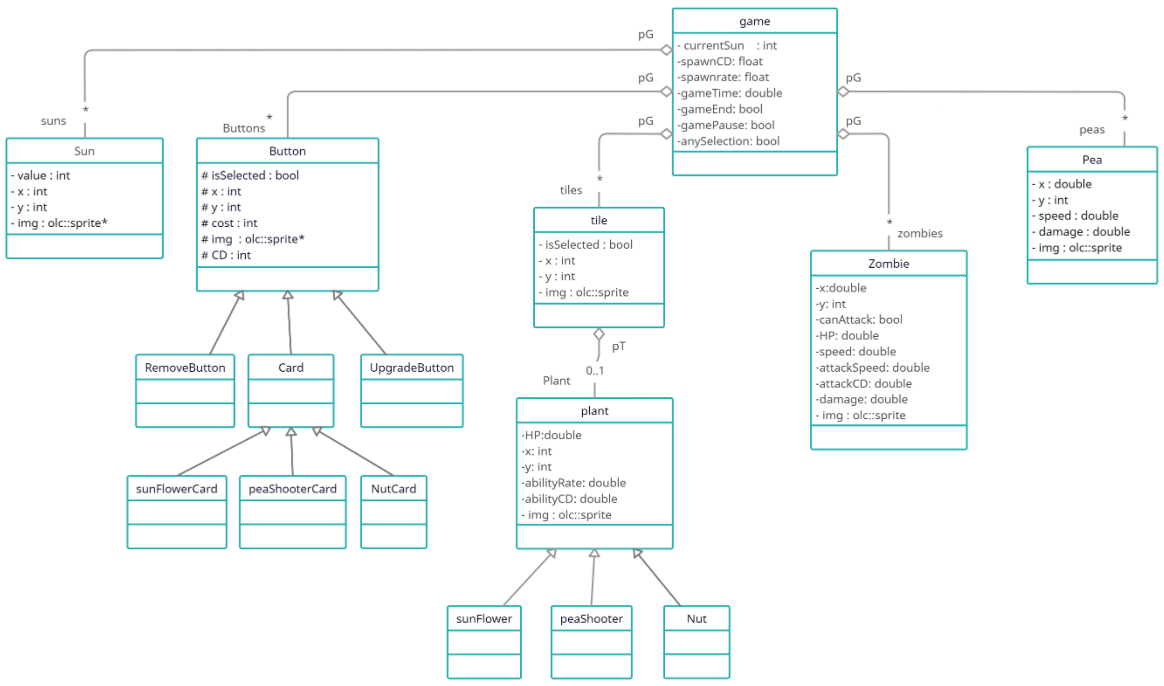
upgradePlant use case allows the player to upgrade a plant after selecting the upgrade button.

Player selects a plant to upgrade and “upgradePlant” message is sent to the game. Game gets the mouse locations and sends a message with input parameters of mouse locations and current sun value to the tile.

Tile compares the mouse locations with its coordinates. If the conditions are satisfied, it sends a message to upgrade button to activate upgrading which returns the cost of the upgrade. Then, tile sends the cost to the game to change the current sun.

If the upgrade button is selected and current sun is sufficient, it tells tile to upgrade its own plant. Tile upgrades its plant. Each type of plant uses the virtual “upgrade” function of the plant to change its image and attack cooldown or health point.

CLASS DIAGRAM



Firstly, game object has button objects within ‘buttonss’ variable, multiple tile objects within ‘tiles’ variable, zero or many zombie objects within ‘zombies’ variable, zero or many sun objects within ‘suns’ variable, and zero or many pea objects within ‘peas’ variable. Pointer to game object is stored as ‘pG’ in these variables. Game object also stores the current value of the sun, the time has been passed since the last use case of the ‘onUserUpdate’, time duration for zombie spawn rate together with the count down for zombie spawn.

Tile class represents the places that plants can be planted. Every tile object has the same image, but different coordinates that correspond to the place of the object on the game screen. Tile objects hold the data which are the existence of a plant in that object and the selection of the object. Every tile has at most one plant and tile object holds this plant within the ‘ownPlant’ variable. Also, every plant holds pointer to its tile with the name ‘pTile’.

Button class has attributes for graphical appearance and selection. Button class inherits three classes: ‘card’, ‘upgradeButton’ and ‘removeButton’. If any plant object is selected after a upgradeButton object is selected, the plant is upgraded. If any plant object is selected after a removeButton object is selected, the plant is removed. Card class also inherits three different classes: ‘sunFlowerCard’, ‘peaShooterCard’ and ‘nutCard’. These objects indicate which plant type will be planted. If any tile object is selected after a card object is selected, a new plant is created within the tile object.

Plant class has attributes for health point, cost, ability rate, and ability cooldown along with graphical attributes similar to other objects. As the name indicates health point holds the remaining health of the plant. Cost is the required value of sun for planting. Ability rate is the required time duration for the occurrence of the plant ability whereas ability cooldown is the remaining time for the occurrence of this plant ability. Plant class inherits three different classes: ‘sunFlower’ ‘Nut’ and ‘peaShooter’. These three classes correspond to different plants with different abilities. sunFlower objects produce sun objects, peaShooter objects shoot pea objects. Nut has more health power than other plants.

Pea objects are the objects that peaShooter objects throw. Every pea object is shot by one peaShooter. However, one peaShooter object throws many peas. Each pea objects have speed and move along in the screen until it reaches the right end of the screen. If it encounters any zombie object, it gives attack damage to the zombie object with the same amount of ‘ad’ variable.

Sun objects are the objects that sunFlower objects create. Every sun object is created by one sunFlower object whereas one sunFlower object can create many suns. Each sun has location, image, and value which is the amount of the sun that will add up to the ‘currentSun’ quantity in the game object.

Zombie class has attributes for health point, speed, attack damage, attack speed, and attack cooldown. Health power holds the remaining health of the zombie object which is quite similar to the plant. Zombie objects move along in the screen with a speed until it reaches the left end of the screen. If it encounters any plant object, it gives attack damage to that plant object with the same amount of ‘ad’ variable within the intervals determined by attack speed. Attack cooldown holds the remaining time for the next attack.