# IE 201 TERM PROJECT
## TECHNICAL DESIGN REPORT

### GROUP 3

**AHMET ÜNLER**
**BATUHAN SARITAŞ**
**MURAT ÖZTÜRK**

**INSTRUCTOR: ALİ TAMER ÜNAL**

# Content

# Brief Explanation of the Game

In this semester, as the registered students we were given the duty to develop and reconstruct a game in an object oriented point of view. In order to construct and redesign the game in a proper way, it is crucial to identify the specifications, attributes and the aims of the game.

The game consists of some different objects. Main idea is to keep on defending from the antagonists of the game. So it is important to understand that there Is not a well-defined objective that needs to be completed in order to "win" the game. Hence, it is important to develop an instinct in the player to keep on playing and try to develop his/her record for the game each and every play.

Main content of the game is consisted of sunflowers, peashooters and the zombies as antagonists. Zombies are spawned in some of the places in the map and as antagonists they aim to destroy the creatures (i.e. sunflowers and peashooters) that are spawned by the player, if they manage to pass the defense line of the player, they eventually defeat the player. However as it can be forecasted, the player needs a defense mechanism for the zombies. Sunflowers and peashooters eventually make up this whole defense mechanism. Although the sunflowers do not practically and physically defense the player and its territories, they grow up the suns, which are the monetary unit of the game. In other words, if a player wants to keep on playing and challenging the zombies, it is crucial for him to plant and harvest sunflowers in order to make a sustainable monetary cycle. In addition to a monetary cycle, a player's first aim is to defend himself/herself from the attacking zombies as mentioned. That issue will be held by the peashooters and the seeds they fire. A peashooter is a defender equipped with seeds (i.e. bullets) that fire towards the zombies, aim to decrease and eventually finish the life points of the zombies and kill them. These two objects, peashooters and sunflowers can be selected from the cards of them by the player and they can be planted onto the tiles of the gaming map which are predefined.

The game primarily consist of the patterns described above. As it was mentioned, the players' main aim is to defend themselves as long as possible versus the antagonist characters, zombies. The duration of the game, in other words success rate of the players, will be determined by the choices and the strategies of the players themselves.
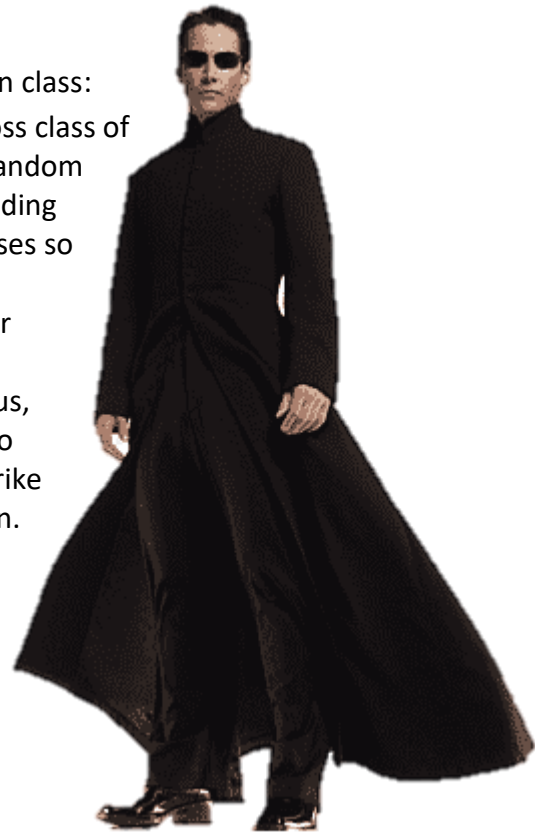
# New additions to the Game

## 1. Matrix Theme

We have converted the theme of the game from plants and zombies to Matrix via changing the names and the sprites of the classes. Zombies will turn into humans who attack peashooters, which will be Agent Smiths in this case. Moreover, Agent Smiths will shoot bullets instead of seeds. Sunflower will not spawn suns, instead they will create energies. Essentially, the will be converted from sunflowers to machines. Via this change in the theme of the game, colors of the tiles and writings in the screens will be subjected to minor changes.

## 2. Neo and Morpheus Classes

We added two new classes to the game which inherit from Human class: Neo and Morpheus respectively. Neo can be considered as the boss class of the game, who can dodge the bullets shot by Agent Smiths by a random chance using a polymorphic Special Power function. Neo's upstanding sprite changes and Y coordinate of Neo, in this maneuver, decreases so that bullets are not able to shoot him. In order to balance the difficulty, the player will not be able to see Neo as often as regular humans. On the other hand, Neo class will be accompanied by Morpheus class which, again, inherits from human class. Morpheus, although as not powerful as Neo, will be using his special power to increase his damage level as his HP drains which enable him to strike Agent Smiths severely by the time he reaches far left of the screen.

## 3. HP Bars

Finally, we added HP Bars to characters that have health level in the game so that player can keep track of the HPs of his agents, machines and humans striking him accompanied by Neos and Morpheuses in gameplay.

# Class Diagram

**olc::Sprite**
...
...

Sprite 1 / Sprite 1
SpecialSprite
SpecialSprite

HumanSprite
MachineSprite
BulletSprite
SmithSprite
EnergySprite
MorpheusSprite
NeoSprite
NeoSprite2
1...*

**Neo**
SpecialActive:bool
SpecialCD:double
SpecialDuration:double

Move(fElapsedTime:float):void
SpecialPower(B:DynamicObject*):void

**Morpheus**

SpecialPower(B:DynamicObject*):void

**PixelGameEngine**

**Plants Game**
gameTime : double
gameEnd : bool
gamePause: bool

anyCardSelected : bool
CurrentSun : int
spawnCD, spawnRate : float
MspawnCD, spawnRate : float
NeospawnCD, spawnRate : float

OnuserUpdate(fElapsedTime:float):bool
OnuserCreate():bool
ObjectUpdate(fElapsedTime:float):void
DrawFrame():void
MouseOperate():void
addEnergy():void
createMachine(x:double,y:double,T:Tile*):void
createSmith(x:double,y:double,T:Tile*):void
createEnergy(x:double,y:double):void
createBullet(x:double,y:double):void
IsEnoughEnergy(Cost:int):int

**Human**
Speed: double
AttackCD: double
AttackRate: double
Damage: double
CanAttack: bool

DidZombiePass():bool
useSpeciality(fElapsedTime:float,
D:DynamicObject*):void
SpecialPower(B:DynamicObject*):virtual void
Move(fElapsedTime:float):virtual void
Draw():void

Humans 1...*
game 1
game 1
game 1
game 1

**Tile**
IsFull: bool

Draw():void
Highlight():void
Plant(C:Card*):void
setEmpty():void
getY():double

Tiles 1...* / TileHabited 1
TileHabited 1

**EnvironmentObject**
x,y : double

IsIn(x:double,y:double):bool
Draw():void

**DynamicObject**
HP : double
x,y : double

lowerHP(damage:double):void
Draw():virtual void
IsClose(x:double,y:double,e:double):bool
useSpeciality(fElapsedTime:float,
D:DynamicObject*):pure virtual void
isErasable():virtual bool
Move(fElapsedTime:float):virtual void
ChangeSprite(S:olc::Sprite*&):void

1...* Plants
game 1
1 game 1 game 1 game

**Card**
IsSelected: bool
Cost: int

Draw():void
Highlight():void
Deselect(anySelection:bool&):void
Select(anySelection:bool&):void
IsPlantable():bool
CreateType(Tx:double,Ty:double,
T:Tile*):pure virtual void

Cards 1...*

**Machine**
EnergyRate: double
EnergyCD: double

useSpeciality(fElapsedTime:float,
D:DynamicObject*):void
Draw():void

**Smith**
AttackCD, AttackRate: double

useSpeciality(fElapsedTime:float,
D:DynamicObject*):void

**Bullet**
Speed: double
Damage: double

useSpeciality(fElapsedTime:float,
D:DynamicObject*):void
IsErasable(fElapsedTime:float):bool
Draw():void
Move(fElapsedTime:float):void

Bullets 1...*

**Energy**
EnergyGain: int

IsIn(x:double,y:double):bool
useSpeciality(fElapsedTime:float,
D:DynamicObject*):void

Energies 1...*

**CardMachine**

CreateType(Tx:double,Ty:double,
T:Tile*)

**CardSmith**

CreateType(Tx:double,Ty:double,
T:Tile*)

# Explanation of Classes and Interrelations

## 1. PlantsGame

PlantsGame is the main class in the game. All the other classes aggregate into this class. It contains the functions which enables user and the CPU to update the gaming environment. It inherits from the PixelGame Engine, which contains the functions that enable the user to start, pause and stop the game. Lastly, Plants Game class has an association relation with the Objects classes.

## 2. Sprite

Sprite is the class that is used to picturize dynamic objects. It aggregates into PlantsGame class and each object of this class holds a specific picture that belongs to one character.

## 3. DynamicObject

DynamicObject class is the parent class of the objects that are created and deleted constantly throughout the game. Its attributes, methods etc. are inherited by other classes such as Smith, Human, Energy since they all have a speciality such as ability of attack or being collected. In addition, DynamicObject has an association relation with PlantsGame class. All character classes which inherit the content of the DynamicObjects class use the attributes separately, freely and independently.

## 4. EnvironmentObject

EnvironmentObject class is the parent class of the objects that are created at the beginning of the game and not deleted until the end. Its attributes, methods etc. are inherited by only Card and Tile. In addition, EnvironmentObject has an association relation with PlantsGame class. All character classes which inherit the content of the EnvironmentObject class use the attributes separately, freely and independently.

## 5. Smith

Smith is the main defense mechanism improved for the player to prevent the harms of the zombies. It shoots Bullets to the humans in order to kill them before they reach and either destruct Smiths or Machines, or directly defeat the player. This class inherits from the DynamicObject.

## 6. Human

Human class is the main enemy of the gaming environment. They occur randomly with a distribution which will be decided by the game developers and their main focus is to harm the player or the gaming contents which are spawned by the player and make player lose the game. They have life points which are defined under this class, and their life would decrease by the Bullets which are threw by the Smiths. This class inherits from the DynamicObject and has an aggregation relation with the class PlantsGame.

## 7. Tile

Tiles are the squares in which the player can deploy the Machines or Smiths in order to continue playing the game. Tiles would eventually be an important aspect for player to build up the strategy for the game. This class inherits from the EnvironmentObject and has an aggregation relation with the class PlantsGame. Also Tile has an association relation with Smith and Machines.

## 8. Bullet

Bullets are the weapons of the Smiths. They are fired by the Smiths and every time they hit their targets, humans, they decrease the life points of the humans. For its properties, Bullet class can be seen as the other part of the defense mechanism for the player. This class inherits from the DynamicObject.

## 10. Machine

Machines are the contents which are deployed by the player to the tiles and spawn energies, the monetary unit of the game which will be held as a class below. In order a player to continue playing the game and earn monetary units of the game, Machines are essential. This class inherits from the DynamicObject.

## 11. Energy

Energies are the monetary units of the game. They are spawned by the Machines, which are deployed by the player. In order to deploy new Smiths or Machines, Energies are essential and needed. Each of the Machines and Smiths will cost some amount of energies, and will be deployed in exchange of this amount. This class inherits from the DynamicObject.

## 12. Card

Card class is the class in which the selection of the Machines or Peashooters are made. Each of these class will have a card in the upper left of the screen and the player would select the one which he/she wants to deploy into the tiles. This class inherits from the EnvironmentObject and has an aggregation relation with the class PlantsGame.

## 13. Neo

Neo class inherits from human class and basically is a human who can also use a special power besides general human powers. He can dodge the bullets shot by Smiths with a random probability. If he succeeds, that particular bullet continues its way through the screen without hurting Neo.

## 14. Morpheus

Morpehus class inherits from human class and basically is a human who can also use a special power besides general human powers. He has a range-like power meaning that as his HP drains, his damage increases.

# Use Case Diagram
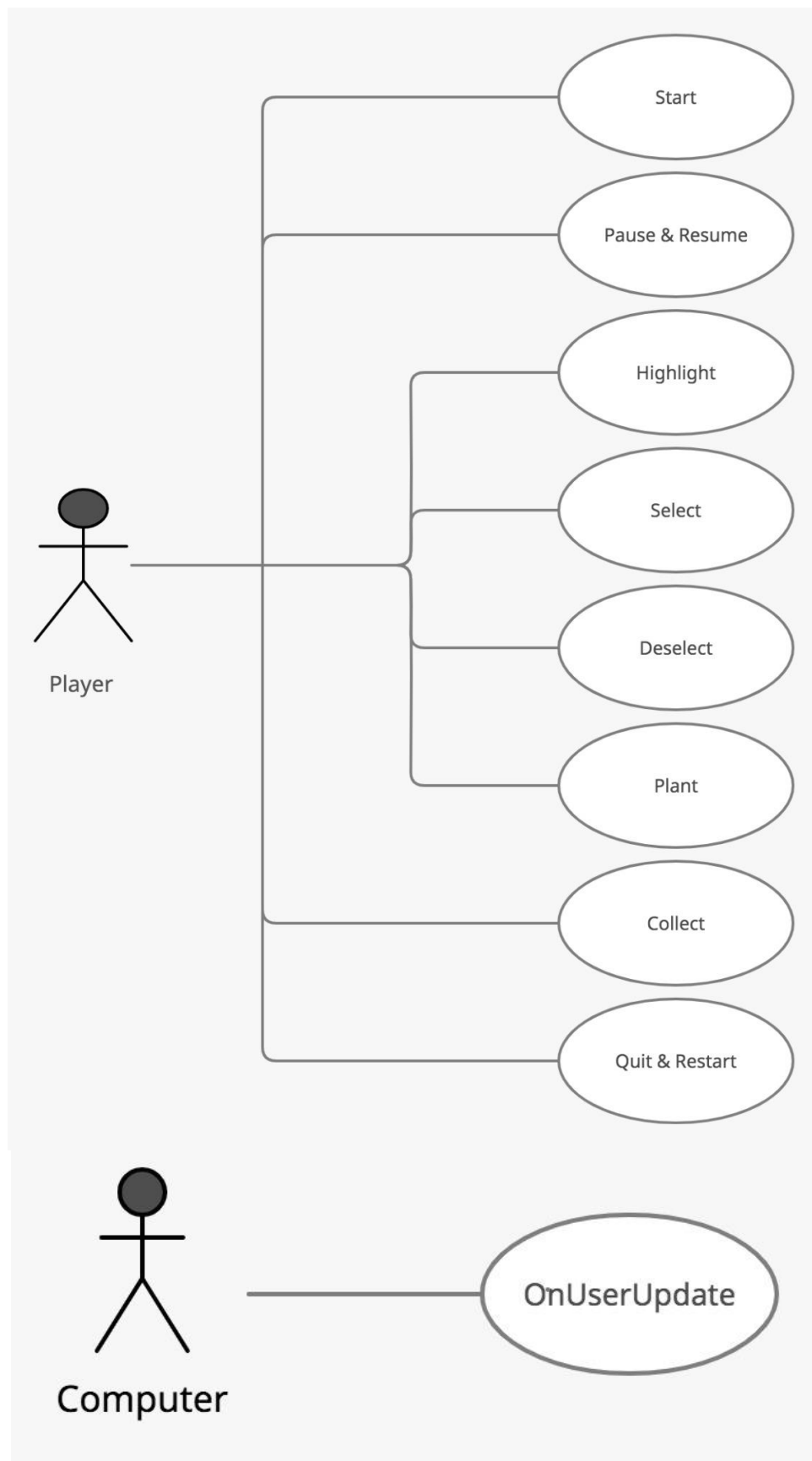
# Use Case Explanation

There are two external agents in this game, which are Computer and Player.

## 1. Start

Game starts from the beginning and game screen pops up when player clicks the button.

## 2. Pause & Resume

Game pauses when user clicks to "P" and a pause screen pops up. User has only one option here which is resuming the game.

## 3. Highlight

When player hovers his/her mouse over the cards or tiles on the game screen, game detects this movement and surrounds the frame of that specific card or tile with a color of white.

## 4. Select

Player can select one of two cards that are shown on the upper left of the game screen. He/she can choose either a Machine or a peashooter to deploy onto tiles.

## 5. Deselect

If the player selects a card which was already selected, the card gets deselected which simply means that the red frame surrounding the card disappears.

## 6. Plant

The player can deploy a plant that was selected onto chosen tile.

## 7. Collect

Player can collect the suns created by Machines via clicking on the suns.

## 8. Quit & Restart

Player can quit the game by clicking "E". After quitting a menu pops up in which player can either leave the game or restart.

## 9. OnUserUpdate

This function is a function of internal agent/ computer. Computer keeps track of time and uses this data to picturize the game screen instantaneously. Each picturization means an iteration and thus, a call to onUserUpdate function.

# COLLABORATION DIAGRAMS AND EXPLANATIONS

## 1-Start

First calls the Construct function which prepares the PixelGameEngine. Then calls OnUserCreate and creates the essential stuff in the game such as Tiles and Cards.
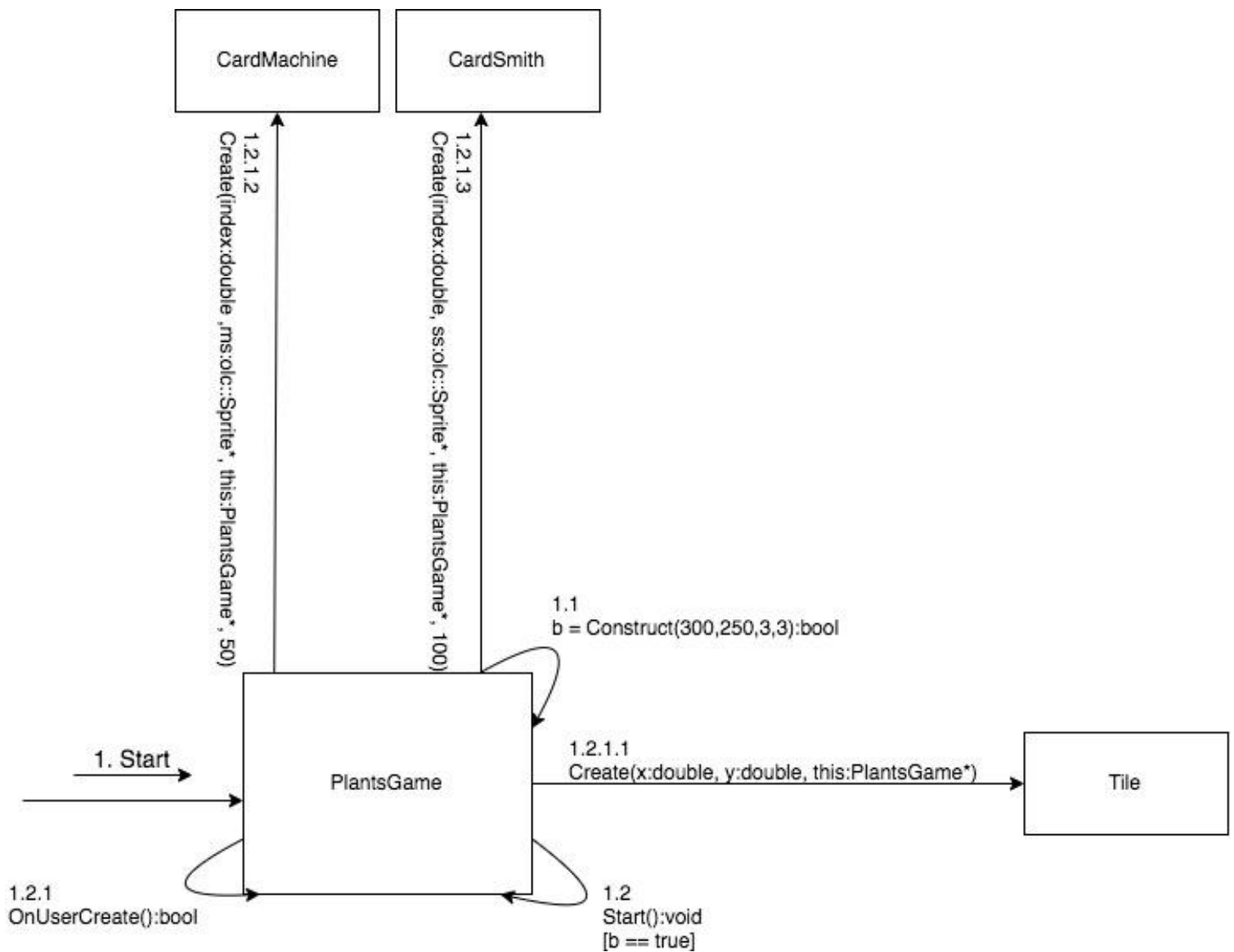
## 2- Pause & Resume

When the key "P" is pressed, Game goes to the Pause screen where it asks itself the positions of the mouse and If Left Mouse is clicked and then it Resumes or stays Paused depending on the conditions

1.2 Clear(olc::BLACK:olc::Pixel):void;
FillRect(100, 100, 100, 50, olc::DARK_GREEN:olc::Pixel):void;
DrawString(90, 10, "PLANTS & ZOMBIES PAUSED", olc::WHITE:olc::Pixel):void;
DrawString(120, 120, "CONTINUE", olc::WHITE:olc::Pixel):void;
SetPixelMode(olc::Pixel::ALPHA:enum):void; [gamePause == true]

1.1 gamePause = true : bool [GetKey(olc::Key::P).bPressed:bool]

1.3 mx = GetMouseX():int32_t

1. Pause & Resume

PlantsGame

1.4 my = GetMouseY():int32_t
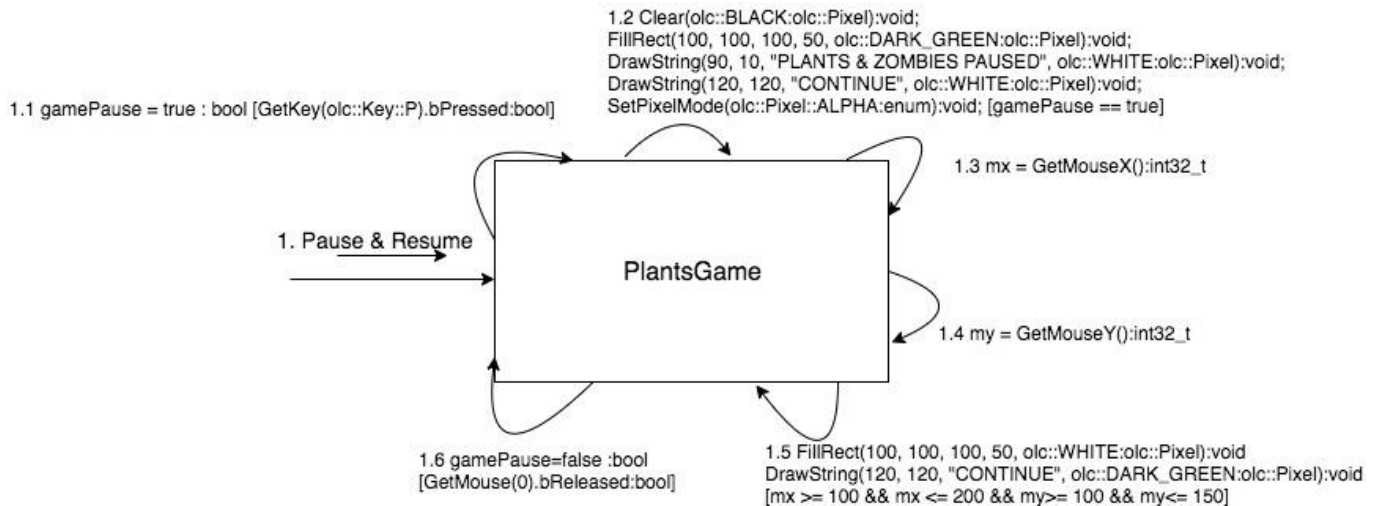
1.6 gamePause=false :bool
[GetMouse(0).bReleased:bool]

1.5 FillRect(100, 100, 100, 50, olc::WHITE:olc::Pixel):void
DrawString(120, 120, "CONTINUE", olc::DARK_GREEN:olc::Pixel):void
[mx >= 100 && mx <= 200 && my>= 100 && my<= 150]

## 3- Select

When left mouse is clicked, Select function goes into process. PlantsGame tells Card to use select function. Card asks PlantsGame if mouse positions are on Card. If so, it also looks if it is not already selected and then if that's the case, it gets selected

1.1.1.3 b = IsIn(x:double, y:double) : bool

1.1.1.1 x = GetMouseX():int32_t

1. Select

PlantsGame

1.1.1.2 y = GetMouseY():int32_t

1.1.1 Select(anyselection:bool):void

Card

1.1 MouseOperate():void

1.1.1.4
if(!IsSelected)
{IsSelected = true;}
[b == true]
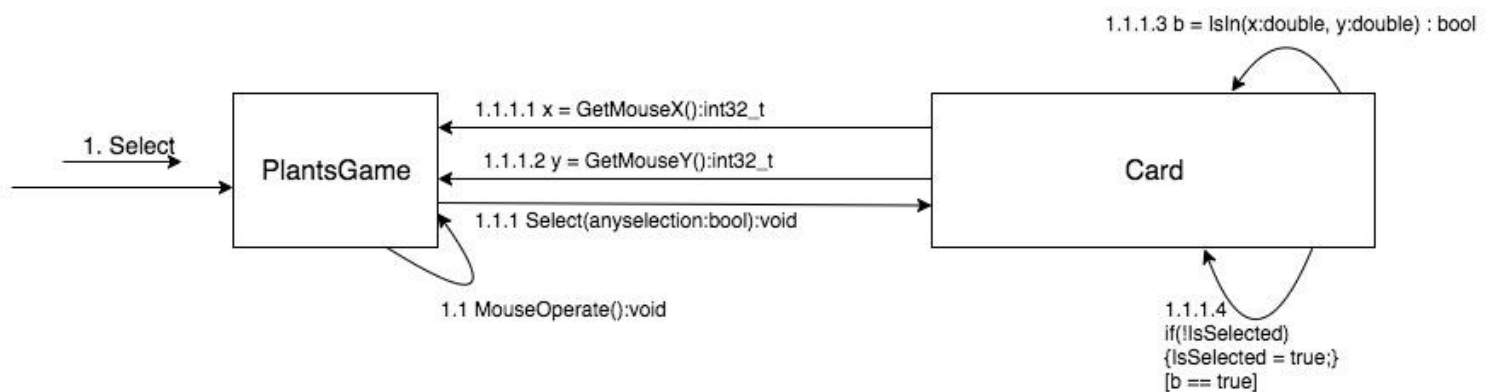
## 4- Deselect

When left mouse is clicked, Deselect function goes into process. PlantsGame tells Card to use Deselect function. Card asks PlantsGame if mouse positions are on Card. If so, it also looks if it is selected and then if it is, it gets deselected.
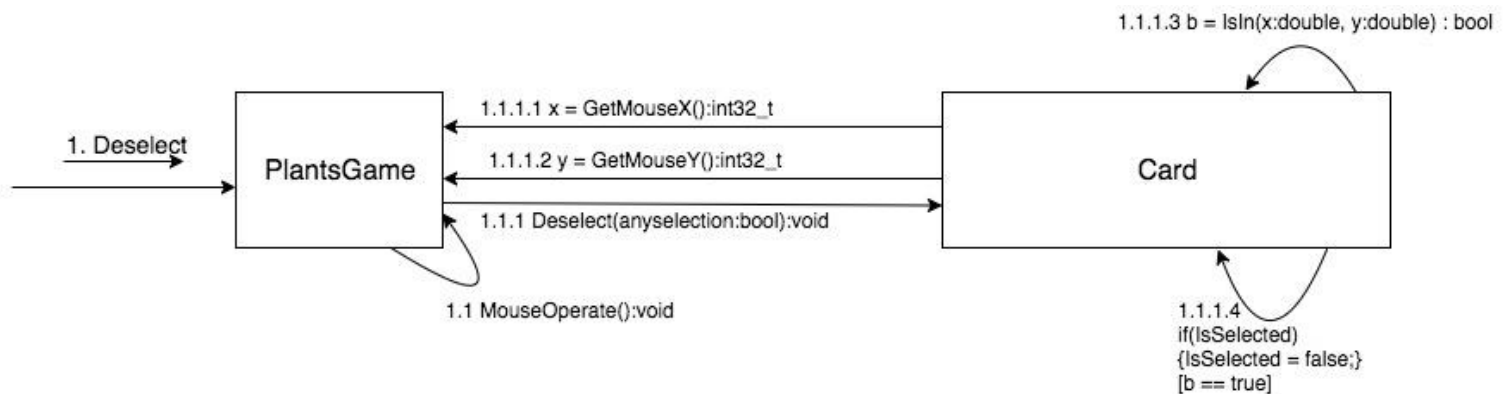


## 5- Collect

When left mouse is clicked, Collect function goes into process. PlantsGame tells Energy to use useSpeciality function. Energy asks PlantsGame if mouse positions are on Energy. If so, Energy gets collected.
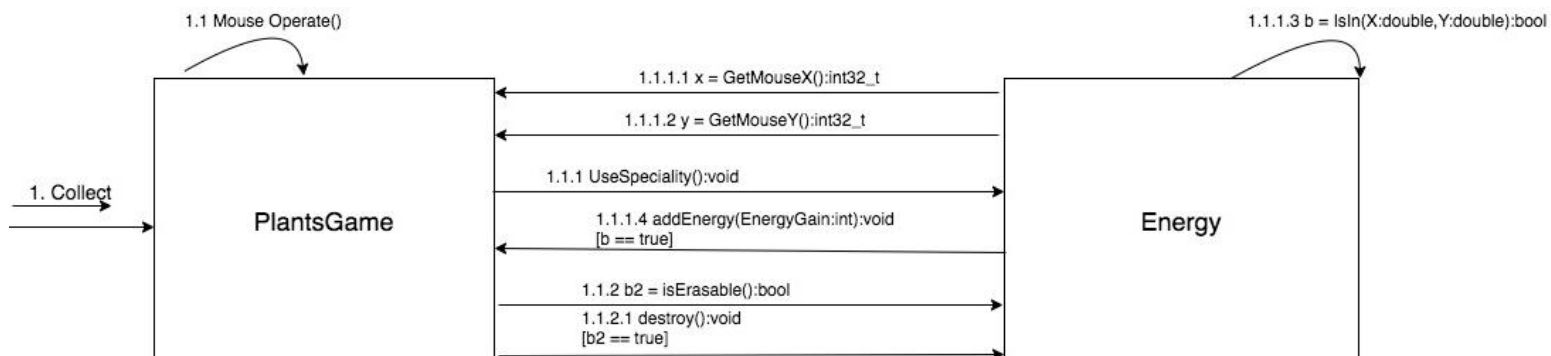
## 6- Plant

Player plants the selected character (Smith or Machine) to the tile of his/her choice. After the game questions whether that particular card is plantable due to cost restrictions or not, it creates a new machine/Smith in the PlantsGame.
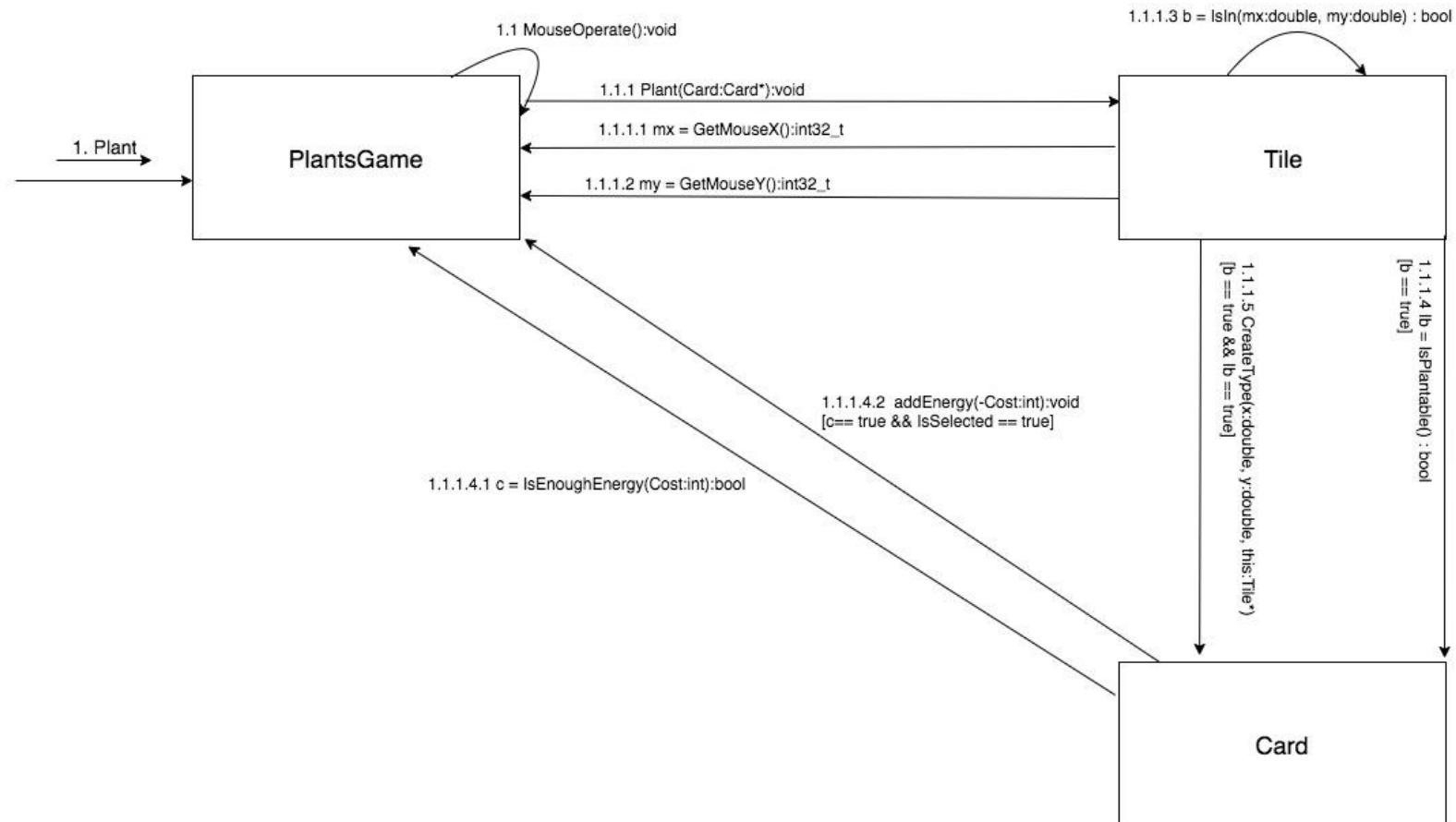
# 7-Highlight

PlantsGame tells Card to use Higlight function. Card asks PlantsGame if mouse positions are on Card . If so it checks some conditions on which colour to highlight the Card, then highlights it.

1.1.1.3
b =IsIn(mx:double,my:double):bool

1.1.1.1 mx = GetMouseX():int32_t

1.1.1.2 my = GetMouseY():int32_t

1.1 MouseOperate():void

1.Highlight

**PlantsGame**

1.1.1 Highlight():void

1.1.1.4 DrawRect(x:double, y:double, 49, 49, olc:: RED:olc::Pixel)
[IsSelected == true]

1.1.1.5
DrawRect(x:double, y:double, 49, 49, olc:: WHITE:olc::Pixel)
[b == true && IsSelected == false]

1.1.1.6
DrawRect(x:double, y:double, 49, 49, olc:: BLACK:olc::Pixel)
[b == false && IsSelected == false]

**Card**

1.1.2.4
DrawRect(x:double, y:double, 49, 49, olc:: WHITE:olc::Pixel)
[b == true]

1.1.2.2 my = GetMouseY():int32_t

1.1.2.1 mx = GetMouseX():int32_t

1.1.2 Highlight():void

1.1.2.5
DrawRect(x:double, y:double, 49, 49, olc:: VERY_DARK_GREY:olc::Pixel)
[b == false]

**Tile**

1.1.2.3
b =IsIn(mx:double,my:double):bool

## 8- Quit & Restart

Player can close the game screen pressing "E" button. After that, PlantsGame destroys all of the objects and presents the player options to close the program permanently or restart the game.



1.1 if(GetKey(olc::E).bPressed)
gameEnd = true;

1.2. DestroyObjects()
[ gameEnd == true]

1.3 mx = GetMouseX():int32_t

1.4 my = GetMouseY():int32_t

1. Quit & Restart

1.7.1 return false
[GetMouse(0).bReleased = true]

1.7 FillRect(100,150,100,50, olc::WHITE: olc::Pixel):void
DrawString(135, 170, "EXIT", olc::DARK_RED::olc::Pixel)
[mx >= 100 && mx <= 200 && my >= 150 && my <= 200]

PlantsGame

1.6.1. OnUserCreate() : bool;
[GetMouse(0).bReleased = true]

1.2.1Destroy()    Human

1.2.2 Destroy()    DynamicObject

1.2.3 Destroy()    Card

1.2.4 Destroy()    Tile

1.2.5 Destroy()    Bullet

1.2.6 Destroy()    Energy

1.6  FillRect(100, 90, 100, 50, olc::WHITE:olc::Pixel) : void;
DrawString(110, 110, "PLAY AGAIN", olc::DARK_GREEN:olc::Pixel) : void;
[mx >= 100 && mx  <= 200 && my >= 90 && my <= 140]

1.5 Clear(olc::BLACK:olc::Pixel):void;
DrawString(110, 10, "GAME OVER", olc::WHITE:olc::Pixel):void
FillRect(100, 90, 100, 50, olc::DARK_GREEN:olc::Pixel):void
DrawString(110, 110, "PLAY AGAIN", olc::WHITE:olc::Pixel):void
FillRect(100, 150, 100, 50, olc::DARK_RED:olc::Pixel):void
DrawString(135, 170, "EXIT", olc::WHITE:olc::Pixel):void
SetPixelMode(olc::Pixel::ALPHA:enum):void
[gameEnd=true]

## 9- OnUserUpdate

It executes operations such as Creating Enemies, calling the essential functions of Objects like telling Agents to shoot Bullets, erasing dead Objects, also Drawing each Object, in order to keep the game running.



1.1ObjectUpdate(fElapsedTime:float):void

1.1.1 x = ScreenWidth():int32_t

1.1.2 Create(x:double,y:double, hs:olc::Sprite*,this:PlantsGame*)

1.1.5 useSpeciality(fElapsedTime:float, target:DynamicObject*):void

1.1.6 move(fElapsedTime:float):void

1.1.7 useSpecialPower(target:DynamicObject*):void

1.1.8 IsErasable():bool

1.2.5 Draw():void

1.3 b = DidHumanPass():bool

1.OnUserUpdate(fElapsedTime:float):bool

Human

1.1.3 Create(x:double, y:double, neo1:olc::Sprite*, neo2:olc::Sprite*, this:PlantsGame*)

Neo

1.4 gameEnd =true; [b == true]

1.1.4 Create(x:double, y:double, ms:olc::Sprite*, this:PlantsGame*)

Morpheus

1.2.6 Draw():void

Energy

PlantsGame

1.1.9 useSpeciality(fElapsedTime:float, target:DynamicObject*):void

1.1.10 move(fElapsedTime:float):void

1.1.11 IsErasable():bool

1.2.4 Draw():void

DynamicObject

1.1.12 useSpeciality(fElapsedTime:float, target:DynamicObject*):void

1.1.13 move(fElapsedTime:float):void

1.1.14 IsErasable():bool

1.2.3 Draw():void

Bullet

1.2 DrawFrame():void

1.2.2 Draw():void
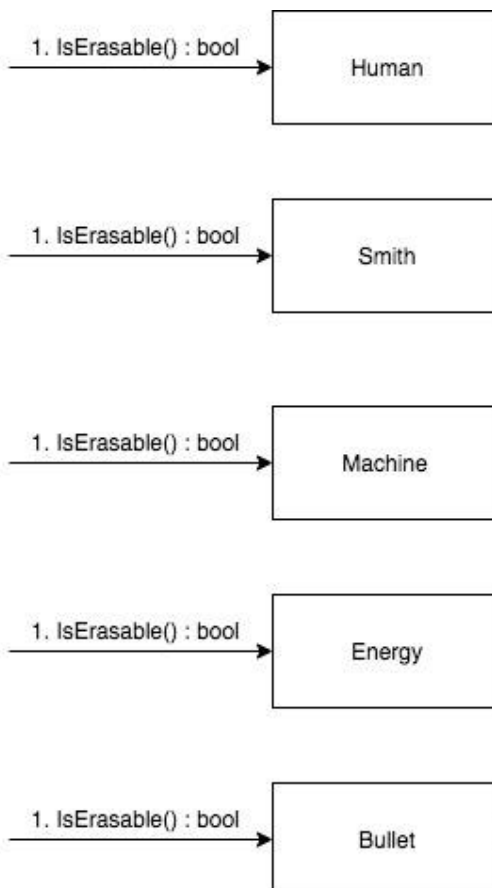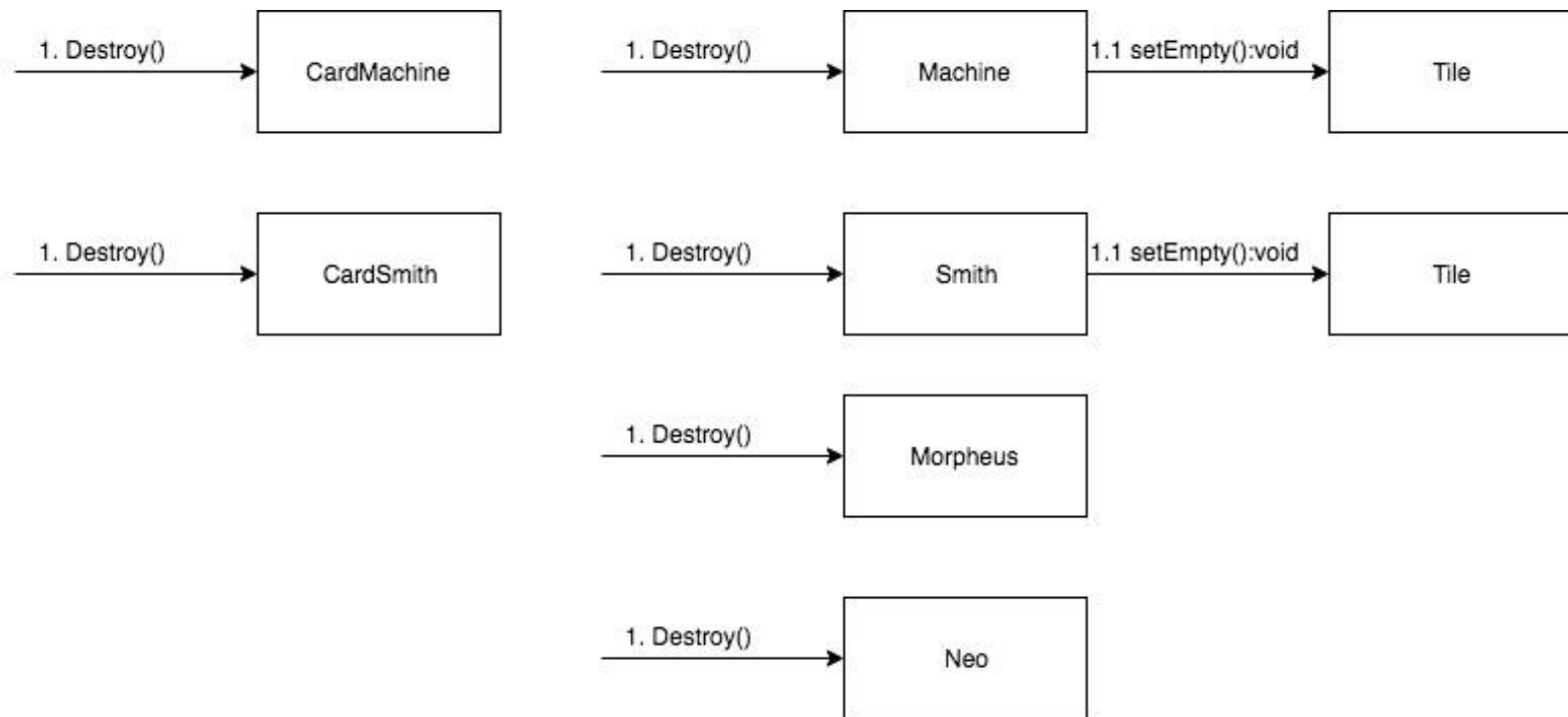
1.2.1 Draw():void

Tile

Card

## 10 – Draw

Draws the object's sprite. Also draws a health bar if it is not Bullet or Energy. Adds cost to card, and draws the borders of tile.
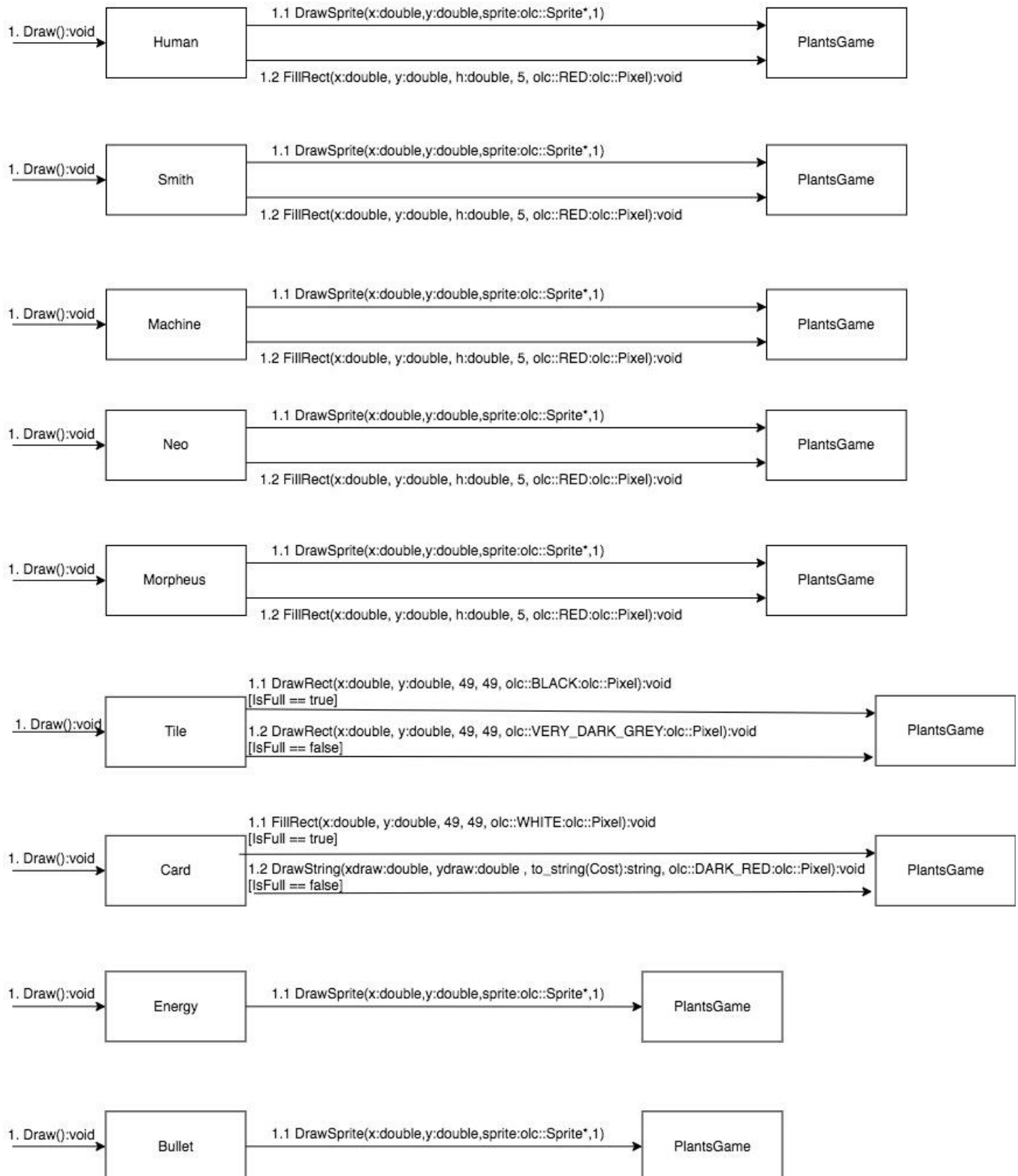
## 11- Destroy

There are three different polymorphic destructors in the game. First one is polymorphic destructor of Cards, then comes that of dynamic objects. Lastly, Morpheus and Neo are destructed via virtual destructor of human.

```
1. Destroy()  →  [ CardMachine ]

1. Destroy()  →  [ CardSmith ]
```

```
1. Destroy()  →  [ Machine ]  1.1 setEmpty():void →  [ Tile ]

1. Destroy()  →  [ Smith ]  1.1 setEmpty():void →  [ Tile ]

1. Destroy()  →  [ Morpheus ]

1. Destroy()  →  [ Neo ]
```

```
1. IsErasable() : bool  →  [ Human ]
```

## 12- IsErasable

Asks DynamicObjects if they should be erased.
Bullet takes ScreenWidth in order to know if it should be erased.

```
1. IsErasable() : bool  →  [ Smith ]

1. IsErasable() : bool  →  [ Machine ]

1. IsErasable() : bool  →  [ Energy ]

1. IsErasable() : bool  →  [ Bullet ]  1.1 ScreenWidth():int32_t →  [ PlantsGame ]
```
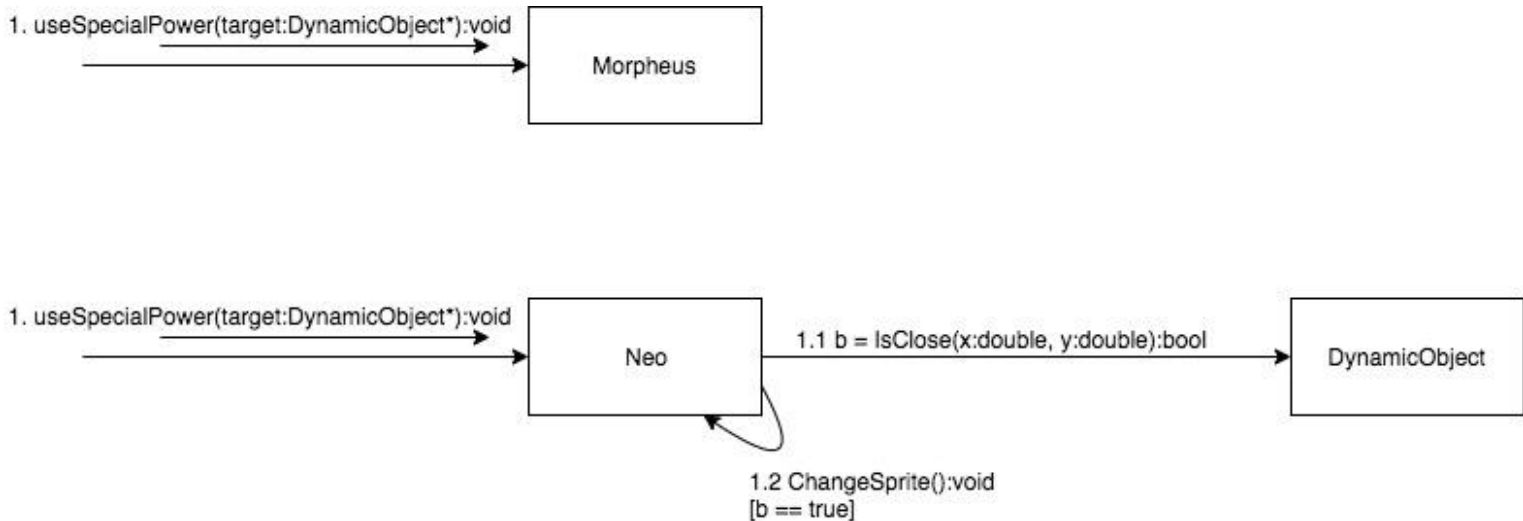
## 13- useSpeciality

useSpeciality function works different for all characters which makes it the core of our polymorphic structure. Machines uses it to create Energies while humans use it to attack plants. On the other hand, Smiths shoots bullet via this function, bullet drains the HP of humans and finally, energy gets collected.
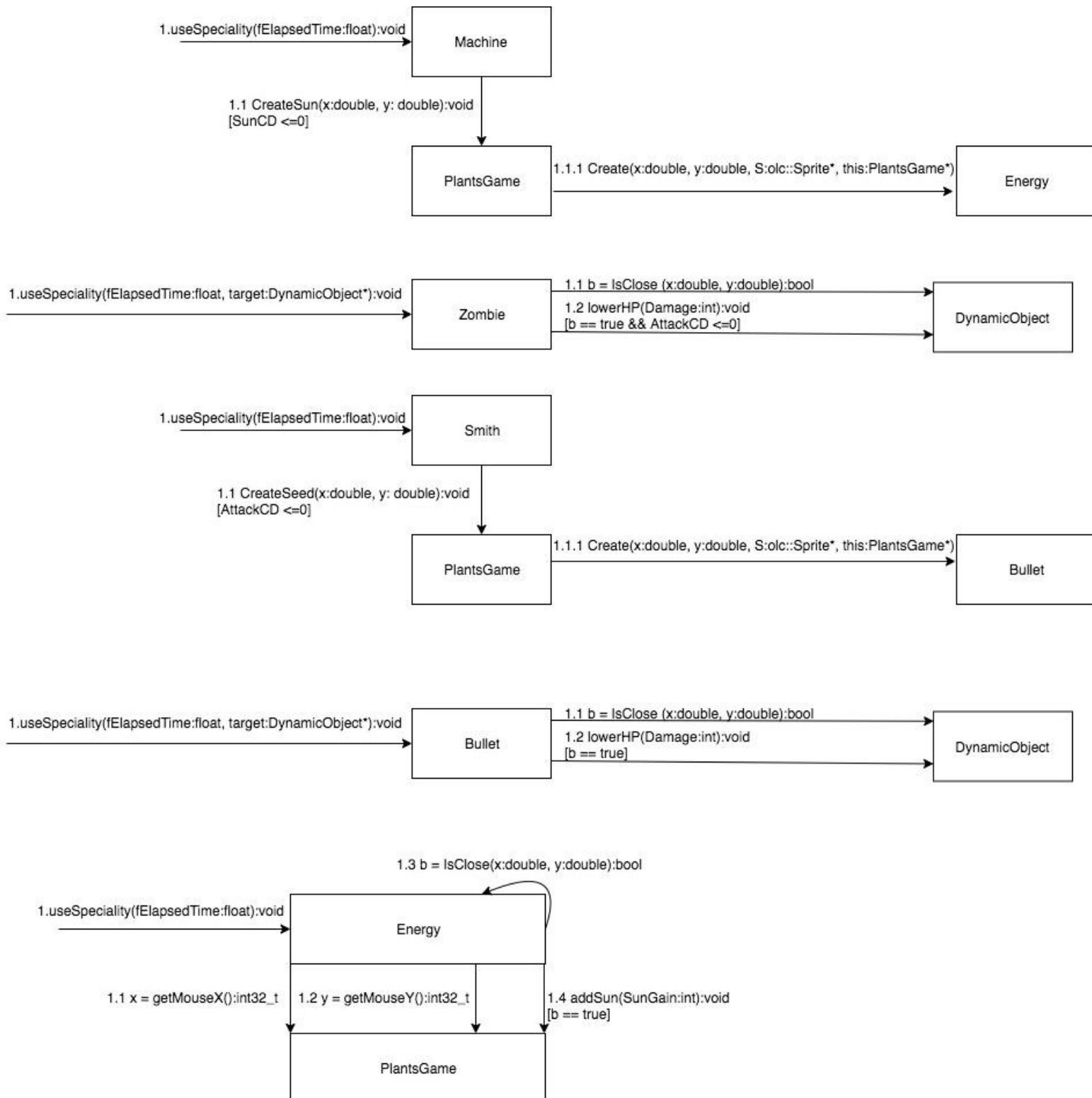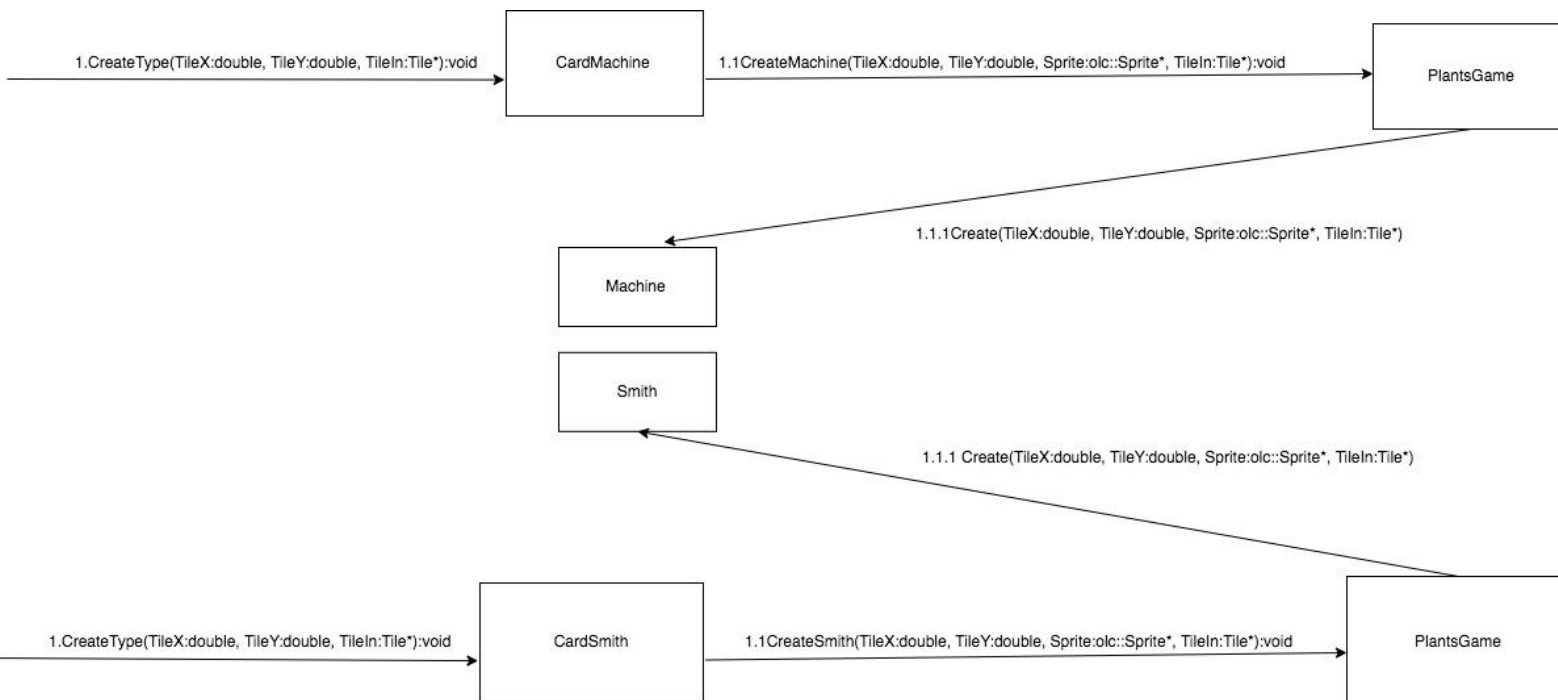
## 14- SpecialPower

This function is a newly added function which is utilized by Neo and Morpheus. Neo uses it to determine whether a bullet is close to him or not and given it is close, he dodges the bullet by a random chance. On the other hand, Morpheus increases his damage as his HP decreases by this function.

1. useSpecialPower(target:DynamicObject*):void → **Morpheus**

1. useSpecialPower(target:DynamicObject*):void → **Neo** → 1.1 b = IsClose(x:double, y:double):bool → **DynamicObject**

1.2 ChangeSprite():void
[b == true]

## 15- CreateType

CreateType is a polymorphic function of Card class which is utilized by CardMachine and CardSmith. As the names suggest, they send a message to PlantsGame to create related characters

1.CreateType(TileX:double, TileY:double, TileIn:Tile*):void → **CardMachine** → 1.1CreateMachine(TileX:double, TileY:double, Sprite:olc::Sprite*, TileIn:Tile*):void → **PlantsGame**

1.1.1Create(TileX:double, TileY:double, Sprite:olc::Sprite*, TileIn:Tile*) → **Machine**

**Smith**

1.1.1 Create(TileX:double, TileY:double, Sprite:olc::Sprite*, TileIn:Tile*)

1.CreateType(TileX:double, TileY:double, TileIn:Tile*):void → **CardSmith** → 1.1CreateSmith(TileX:double, TileY:double, Sprite:olc::Sprite*, TileIn:Tile*):void → **PlantsGame**

## 16- Move

Bullet and Human move via the polymorphic function written in Dynamic Object. On the other hand, Morpheus and Neo use the polymorphic move function of Human. Neo checks whether he should move or dodge the bullet depending on his relative location to the bullet.

1. Move(fElapsedTime:float):void → **Bullet**

1. Move(fElapsedTime:float):void → **Human**

1. Move(fElapsedTime:float):void → **Morpheus**

1. Move(fElapsedTime:float):void → **Neo**

1.1 Move(fElapsedTime:float):void
[SpecialActive == false]

1.2 ChangeSprite(SpecialSprite:olc::Sprite*):void
[SpecialActive == true && SpecialDuration <= o]