Controlling a charged particle with a genetic algorithm

Tomoyuki Jinno
PH2150 Scientific computing report No.1

December 13, 2018

Abstract

A charged particle can be controlled by manipulating electric field around the particle. Finding a theoretically optimal solution to control a particle in a desired path under influence of undesired electric and magnetic field can be difficult. Genetic algorithm can be useful when finding an optimal solution in a system that is affected by many variables. I investigated possible application of genetic algorithm in controlling a charged particle with 3 charged plates that are at right angle to each other.

1 Introduction

The goal of this project is to write a computer program that can control a charged particle in a specific path. The program consist of two tabs. "Environment builder" tab is used to specify the intended path of the particle and the external electric and magnetic field that influence the particle. "Simulation" tab is used to train the genetic algorithm and observe the path of particle controlled by the algorithm. It is important to note that all simulation of electric and magnetic field and the motion of the particle used in this project do not take quantum mechanical effects into count.

1.1 Set-up of the simulation world

The simulation world contain three charged plates that are placed in right angle to each other. These plates will be referred to as control plates. Electric potential across the control plates can be changed over time to control the electric field around the charged particle. A series of three dimensional vector can be given to change the pd across the control plates with respect to time. This series of vector will be referred to as control tape. Each row in the control tape represent the potential difference of a control plate at a specific time. The E field strength across the control field will act as if the separation of the plates is 1m even though the range of this E field will be simulated to be infinite. This set-up along with obstacles that contribute to the electric and magnetic field is referred to through out the project as simulation world.

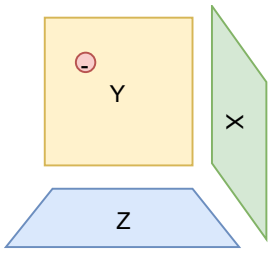


Figure 1: Diagram representing control plates with charged particle

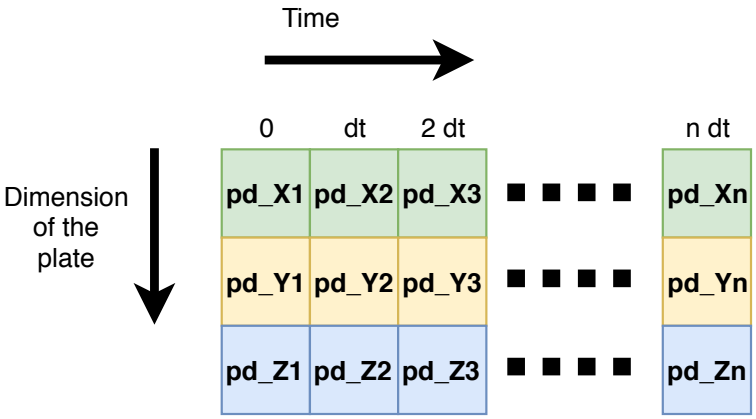


Figure 2: Diagram representing a control tape

1.2 Genetic algorithm

Genetic algorithm is a type of optimization algorithm inspired by the natural selection concept in Darwen's theory of evolution. The algorithm can be generally be split into 3 parts.

1. Selection: Species with high "fitness" are selected to influence the genetic information of next generation.
2. Evolution: New set of species are made from the genetic information of the selected species.
3. Mutation: Genetic information of the new set of species are varied randomly.

The implementation of these concepts can vary greatly depending on the application and desired number of generations taken to find the optimal solution.

Genetic algorithm is a reinforcement learning model, though it may depend on the implementation. Reinforcement learning models are learning models that are exposed into an environment with constraint and goal to maximize it's fitness. The model will attempt to find an optimal solution by experimenting through trial and error method in the exposed environment. This is one of the advantages of a reinforcement learning model since it does not require any training data to train the model.

In this project, the environment the genetic algorithm is exposed to is the simulation world and the model is deemed fit when the particle is moved in a desired path specified by the user. Control tape is the equivalence of the genetic information and will sometimes be referred to as the gene or a specie in this project. As for this project, a collective of genes or species is called population and a collection of population is called generations.

1.3 Object oriented programming principles

This project have heavy use object oriented programming paradigm, therefore key concepts of oriented programming paradigm used in the project are introduced here.

1.3.1 Class relations

Two types of class relations used frequently in this project are as follow.

1. **Association**: Association defines a logical relation between classes. There are two main types of association.
   (a) **Aggregation association**: when two classes do not have a strong life-cycle dependency to each other
   (b) **Composite association**: when one of the classes have strong life-cycle dependency on other
2. **Polymorphism**: when classes have methods that share the same name across a group of classes that represent the same concept,input and return value but operate under different logic that is encapsulated away.
   (a) **Inheritance**: is the method of polymorphism in most programming languages. It allow child class to communicate with parent class in a special way and allow override of methods.

1.3.2 Specifiers

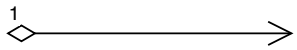
Specifiers encapsulate methods and attributes which contributes to the concept of information hiding. This allow program to be expressed in conceptual logic rather than processes. There are three main types of specifiers.

1. **Public**: Can be accessed from anywhere. I.e Inside class, child class, outside the class.
2. **Private**: Can only be accessed from inside the class and is not accessible to anywhere including it's child class. In Python, private methods start with double underscore e.g. __hello, not to be confused with double double underscore methods (dunder methods) e.g. __hello__ which is a different thing.
3. **Protected**: Definition of protected specifier vary between programming languages. In Python, these are not accessible from outside the classes but is accessible to the child classes when the class is inherited. In Python, protected methods/variables are not enforced by the language but are semantically denoted by a single underscore e.g._hello.

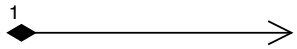
1.3.3 UML class diagrams

UML, Unified Modelling Language, is a standard language used to model entities and relations. They can be used to visually represent classes and their relations. Some of the shapes used to represent class relations in a UML language are as follow.

1. Class: a class is represented by a rectangle with a segment for attribute name and a segment for method names.
2. Inheritance: is represented by an arrow with white tales. The arrow is pointed at the parent class from the child class.
3. Aggregation association: is represented with a line between classes. The line is featured with a white rhombus at the parent class end of the line and an arrow at the other end pointing at it's child class.



4. Composite association: is represented in a similar way as the aggregation association but with a black rhombus rather than a white rhombus.



5. Specifiers: are represented by a character
   (a) Public: +
   (b) Private: -
   (c) Protected: #
6. Attribute: is represented by concatenation of the specifier, attribute name and it's data type. The name and type are separated by a colon E.g. +hello:int
7. Method: is represented by concatenation of specifier,method name, method's arguments and their type and return data type. E.g. +hello(name:string):string

2 Project structure

I will encourage readers to skim read Fig 14 which is a diagram of association relation of the project in order to have a better feel of how the project is structured.

2.1 Class structure

There are five main domains of classes hierarchy in the project that are grouped by their conceptual relations.

Name	Location
gui	guiClean.py
world plotters	worldPlotter.py
objects	object.py
electro magnetic simulation	emSimulation.py
evolution	evolution.py

Table 1: A list of class hierarchies

2.1.1 gui

This project use the Tkinter library to construct a graphical user interface. gui classes are generally responsible for managing widgets to render and responding to input calls from the user. The main classes of gui domain are MainWindow, SimulationTab and BuilderTab. To reduce information, only important attributes and methods are shown in the UML diagram. buildWOrld class have a powerful system that allow users to select and move objects by drag and drop. It is an intuitive user interface to interact with the build world. This mechanism can be represented as a finite state automata represented in Fig 3.

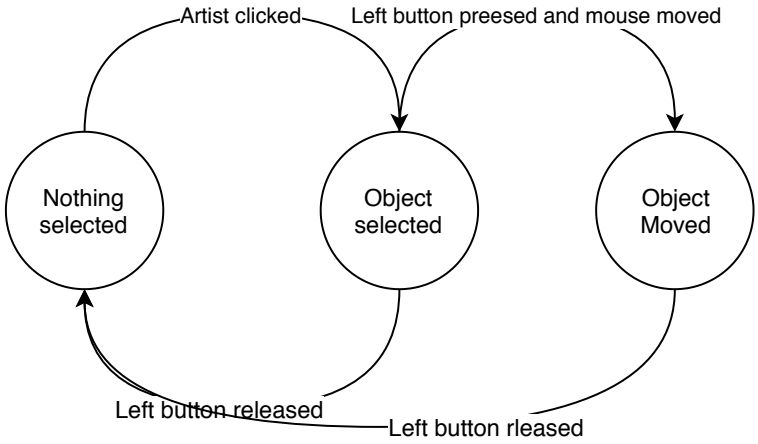


Figure 3: Finite state automata representing the drag and drop mechanism of buildWorld class

2.1.2 world plotters

These set of classes are mainly responsible for plotting the world objects onto figures. They have an association relation with the objectWorld class and will respond to changes made and render the world accordingly. There is a strong use of polymorphism in these classes.

```
def __showForTheFirstTime(self,obj):
    # first time to plot
    COLOR_BY_TYPE={"Start Point":"green","Finish Point":"orange",
                   "Positive Enforcement Token":"gray",
                   "Negative Enforcement Token":"black"}
    artist=None
    objType=obj.type
    if(objType in self.POINT_TYPE):
        position=obj.getNodes()
        artist=self._plotPoint(position)
    elif(objType in self.SURFACE_TYPE):
        surfaces=obj.getSurfaces()
        #choosing color
        if(objType in COLOR_BY_TYPE):
            color=COLOR_BY_TYPE[objType]
        else:
            #default color
            color="blue"
        artist=self._plotSurfaces(surfaces,color=color)
    #updating dictionaries
    if(artist!=None):
        nodes=obj.getNodes()
        self.__appendNewElementToLists(obj,artist,nodes)
    else:
        print("Have not been told how to plot")
```

showForTheFIrstTime is a method of WorldPlotter class. It is called when a new object have been discovered in the objectWorld and it make great use of polymorphism. Object class that is passed into as input may be any child class of the object class, but since all child classes of object class inherit from the object class and override, type attribute, getNodes method and getSurfaces method. This prevents the logic of how nodes and surfaces can be obtained from different type of classes to be abstracted away from the method and treat them with the unified concept of getting the node and getting the surface.

Another form of polymorphism is used in this method. The plotPoint and plotSurfaces methods are called by this method. However, WorldPlotter class does not contain any logic inside plotPoint or plotSurfaces. plotPoint and plotSurfaces methods are overrided by it's child classes, WorldPlot3d and WorldPlot2d. This method defines the conceptual logic, and leave the logic of how plotPoint and plotSurfaces is done to be defined by it's child classes WorldPlot3d and WorldPlot2d. This allow conceptual logic to be shared between the child particle whilst keeping the process separate. This remove redundancy of the same logic from being repeated in WorldPlot2d and WorldPlot3d. Detail of overrided methods in world plotters can be seen in Fig 16.

2.1.3 Objects

Objects consists of classes that can be placed in a world. Obstetrical objects uses polymorphism where all child class of obstacles have getE method and getB method that return E and B value that the obstacle contribute to the position.

2.1.4 Electro magnetic simulation

Electro magnetic simulation consist of classes that are responsible for calculating the electric and magnetic field at various point and in simulating the trajectory of charged particle. These class uses Runge kutta method to achieve this.

2.1.5 evolution

Evolution class is responsible for managing the genetic algorithm.

3 Hands-on tutorial

This section will focus on explaining how to use the program through an example. In this section I will explain how the program can be used to train a model that move a charged particle from start point to finish point across an external uniform electric field.

3.1 Building the world

Environment builder tab can be used to construct the environment the genetic algorithm will be exposed to. When environment builder tab is selected you should observe an equivalent of fig (6).

3.1.1 Figures

There are 4 figures in the tab that render the same world from different perspective. The world that they render will be referred to as build world.

Figure on top left renders the 3 dimensional representation of the build world. This figure can be rotated by left-clicking and dragging the mouse and it can zoom in and out by right-clicking and dragging. 3 dimensional figure is useful for having a grasp of the build world but objects inside the build world can not be selected from the 3 dimensional figure.

There are three 2 dimensional figures in the tab. They are 2 dimensional representation of the build world from different perspective. Figure on top right is viewing the world from top view, bottom left is front view and bottom right is the right view. The two dimensional figures can be used to select and move objects that are in build world by drag and drop.

When the environment build tab is selected for the first time after launch of the program, you should observe an orange cube inside build world. If you move this cube by selecting and dragging on it in one of the 2 dimensional figures, a green cube should be revealed. Position of the green cube defines the starting position of the particle to be controlled by the genetic

algorithm and the orange cube defines the position of desired goal to be aimed for by the genetic algorithm.

3.1.2 Object palette

Object palette can be used to add objects into the build world. Object palette is the section of the tab where there is an add button along with names of various objects in a list box. A object can be added by selecting the name of desired object on the list box and clicking the add button. As part of the tutorial, select on the charged plates and add it to the build world.

3.1.3 Field viewer

Field viewer can be used to observe the electric and magnetic field in the build world. It is section of the world under Field view option text. If you select the E field option, you will immediately observe vertical lines that are pointing downward. These lines represent the direction of the electric field but they do not represent the magnitude, since the lines are normalized to be at constant length. If the objects inside the build world are moved, the field viewer should automatically respond and update it's vector field plot. Field viewer can also be used to observe super imposed electric field as shown in Fig 4

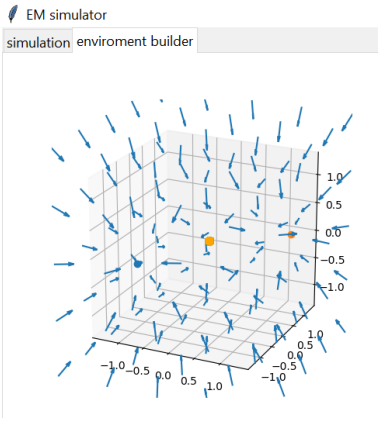


Figure 4: Super imposed electric field of two charged particles

3.1.4 Enforcement tokens

Enforcement tokens can be used to award or penalize species that collect them before reaching the finish position by adding or subtracting from their fitness points. Due to the nature of genetic algorithm, these can be used to motivate the species to move across specific path. Positive enforcement token can only be collected once by a specie to prevent species from violating the constraint by repeatedly passing through a same token. However, the same negative enforcement token can be collected repeatedly this is done to prevent the species from sacrificing some fitness points to move across undesired path. To follow this tutorial, make a triangle outside influence of plate charge using 9 positive enforcement tokens then place the start token at the narrow end of the triangle and position finish token at the other end of the triangle. The build world should look as the one in Fig 5.

3.2 Training the model

Simulation tab can be used to train the genetic algorithm.

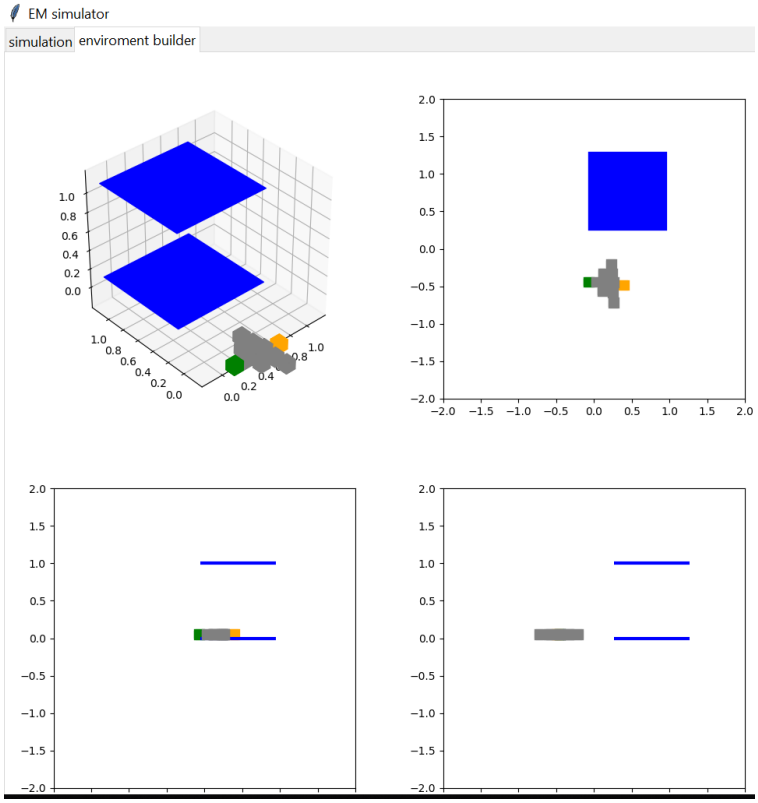


Figure 5: Build world that is constructed in the tutorial

3.2.1 Uploading the build world

Upload button can be used to copy the world constructed in the buildworld to the simulation world. To follow the tutorial, click on the upload build world button, you should observe that the build world is shown in the simulation world figure as shown in Fig 7.

3.2.2 Next generation button

Next generation button can be used to train the genetic algorithm and generate proceeding generation of species. The computation of training can take some time, unfortunately the processing thread of the training and tkinter are on the same. Therefore it is a serial thread computing. This can make the GUI unresponsive and the operating system may decide to kill the program, for this reason, do not click on the GUI whilst the training is in progress. To follow the tutorial, click on train next 10 generations button. This training should take about 2 minutes and 30 seconds.

3.3 Viewing species

Once the training is done, select on generation 0 and select species in the species list box. A black stripe should be appended to the right of the list box. This is the control tape of specie represented as image. As a particular pattern of control tape become more common, you may observe some pattern in the control tape viewer. On top of the control viewer is the fitness viewer where fitness level of the specie is shown. Compare the fitness level of species in generation 0 and species in generation 9, there should be a noticeable improvement in fitness over generations. Simulate control tape button can be used to observe the path of the partial controlled by the specie have taken. As fitness level increase, the path of the particle should tend towards the path of positive enforcement tokens. With some luck, the particle may path through

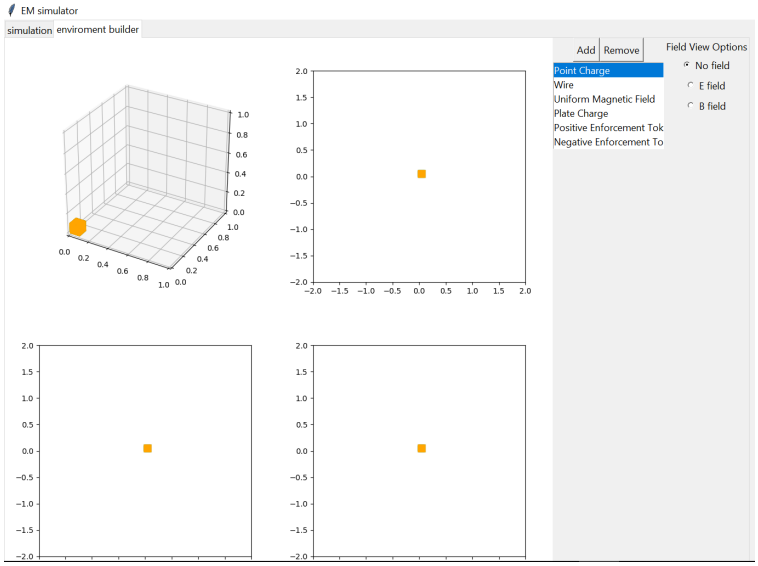


Figure 6: Initial state of the environment builder

area where electric field of the obstacle plate is pleasant, where you may observe the point charge change it's course as it reach the influential area.

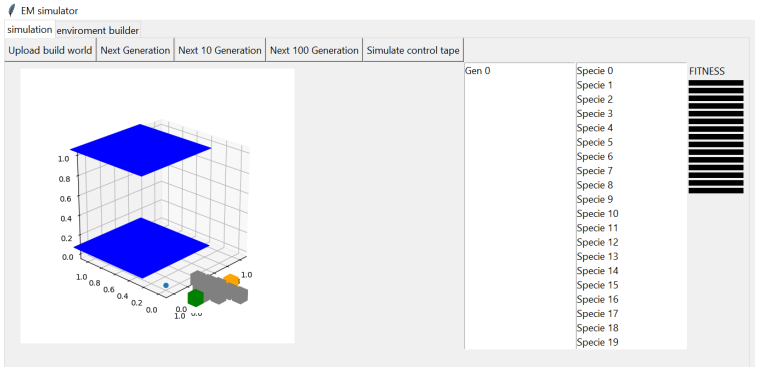


Figure 7: State of simulation tab after the build world is uploaded

4 Implementation of algorithm

Many different algorithms were implemented in the project to achieve desired effects. Here, implementation of notable algorithms are explained.

4.1 Genetic algorithm

Basic concept of genetic algorithm is described in the introduction. This subsection will explain how the concepts were implemented into the program. After the instantiation of EvolutionChamber class, the first method called when training the genetic algorithm is the epoch method.

```
def epoch(self):
    currentPopulation=self.populationOverGenerations[-1]
    populationResults=self.examinePopulation(currentPopulation)
    populationFitness=self.assignFitnessToResults(populationResults)
    numberOfMates=100*2
    matingSpeciesIndex=self.selectingMatingSpecies(
        populationFitness,numberOfMates)
    matingPairsIndex=self.assignPairs(matingSpeciesIndex)
    newPopulation=self.crossOverSpecies(currentPopulation,matingPairsIndex)
    newMutatedPopulation=self.mutatePopulation(newPopulation,self.mutationRate)
    self.populationOverGenerations.append(newMutatedPopulation)
```

The flow chart below shows that the code is equivalent to the three concepts of genetic algorithm.

examinePopulation
↓
assignFitnessToResults } Selection
↓
selectMatingSpecies
↓
assignPairs
↓ } Evolution
crossOverSpecies
↓
mutatePopulation } Mutation

Brief explanation of the methods

examinePopulation This method is responsible for setting the control tape to an emChamber instance, running the simulation and retreaving result of the simulation from the emChamber instance. The result returned from the method is a dictionary that contain the number of positive enforcement tokens collected, the number of negative enforcement tokens collected and boolean on whether the goal was reached by the particle or not.

assignFitnessToResults This method is responsible for weighing the results to create fitness. The equation used to calculate the fitness in this project is as shown below.

$$\text{Fitness} = 1 + \text{posToken} \times 100 - \text{negToken} \times 100 + \text{goalReached} \times 300 \quad (1)$$

This method could be modified to optimize different parameters such as energy used or time taken. A continuous change of incentive to vary the fitness may prove more effective than a discrete incentive such as the positive reinforcement token. The reason for this is because even when a particle pass close by a token, the fitness does not change. However, when the token is collected, the fitness is suddenly increased meaning there is a spike in the rate of change of fitness. This would cause a sudden favouring of particular specie causing discrete breakthroughs rather than a continuous evolution. This system fails to reward small improvement made through evolution and mutation preventing them from being selected as parents. Therefore this discrete rewarding system can not improve the genetic algorithm through accumulation of small improvements.

selectMatingSpecies This method take fitness of the species as input and return the species to paired into couples. There are three parts to this method.

1. **Normalize fitness**
   Fitness of a species are normalised using L2 normalization. L2 normalization is usually chosen over L1 normalization to express anomalies. This is because sum of square of L2 normalization is equal to one, where as in L1 normalization it is sum of the normalized value that is equal to 1. This means that in L2-norm, the difference is more pronounced since they are squared.

2. **Distribute lottery ticket**
   Lottery ticket is a pair of numbers where the numbers between the pair of numbers is unique to the specie. If the lottery draw number is in-between the two numbers, the specie is selected as a parent. Since the probability of being selected as parent should correlate to it's fitness. The range between the the pair of numbers can be given by range = $\text{normedFitness}^2 \times (\text{maxDrawNumber} - \text{minDrawNumber})$

3. **Generate a lottery Draw Number**
   A number between max draw number and min draw number is generated at random. Specie that own a lottery ticket where the draw number is between the pair is selected to be a parent.

selectMatingSpecies This method take fitness of the species as input and return the species to paired into couples. There are three parts to this method.

1. **Normalize fitness**
   Fitness of a species are normalised using L2 normalization. L2 normalization is usually chosen over L1 normalization to express anomalies. This is because sum of square of L2 normalization is equal to one, where as in L1 normalization it is sum of the normalized value that is equal to 1. This means that in L2-norm, the difference is more pronounced since they are squared.

2. **Distribute lottery ticket**
   Lottery ticket is a pair of numbers where the numbers between the pair of numbers is unique to the specie. If the lottery draw number is in-between the two numbers, the specie is selected as a parent. Since the probability of being selected as parent should correlate to it's fitness. The range between the the pair of numbers can be given by range = $\text{normedFitness}^2 \times (\text{maxDrawNumber} - \text{minDrawNumber})$

3. **Generate a lottery Draw Number**
   A number between max draw number and min draw number is generated at random. Specie that own a lottery ticket where the draw number is between the pair is selected to be a parent.

assignPair This method is used to assign species to a pair. Species that are selected as parents are placed in a waiting list until it is paired with another specie. It is possible to have a pair where both species are the same since it is possible for the same specie to be selected multiple time as a parent.

crossOverSpecies This method produce a child gene by crossing over the two parent gene at random.

1. Choose number of segments The child gene have the same length as it's parent's gene. When child gene is initially defined, it all the genetic material is empty. The gene is split into segments where minimum is 2 and maximum is 8 segments.

2. Choose length of each segments The length of each segments are chosen, the reason why the length and segments are used to randomly exchange generic material at random position is because the series of genetic material may contain contextual information such as "move towards right at start", and this contextual information would be lost if the genetic information are scrambled randomly.

3. Pick a parent Chose the parent to base the segment of genetic information on

4. Slot in parent gent to the child gene at the segment Substitute

5. Repeat step 3 and 4 until all segments are filled.

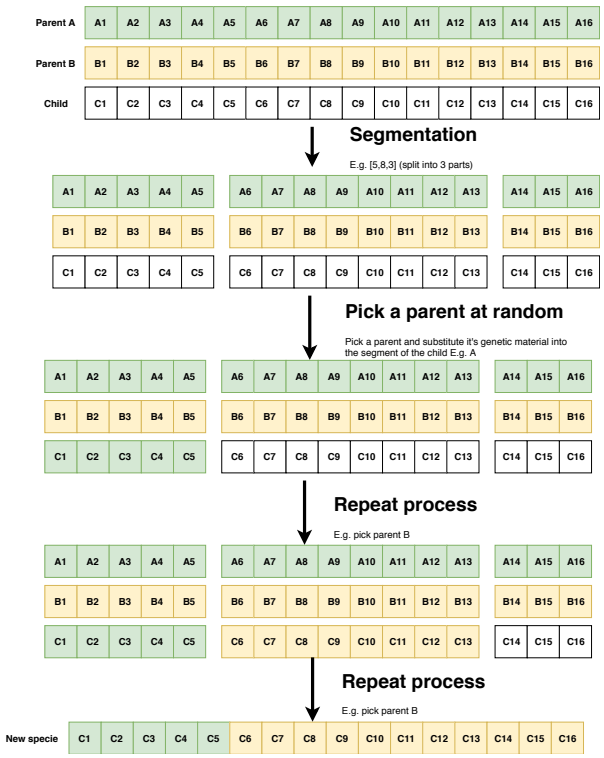


Figure 8: Diagram that represent implementation of crossing over in control tape

mutatePopulation This method will randomly randomize genetic material of species and return them. The mutation rate is set to 10% therefore, each genetic material of the specie have 10% chance of being replaced with a random genetic material. This is done to allow the genetic algorithm to explore new approach to the solution and can prevent the system from getting stuck in a local minimum.

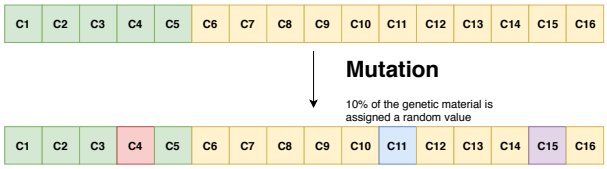


Figure 9: Diagram representing mutation process

4.2 Separating axis theorem

Separation axis theorem is a theorem that can be used to make a collision detection algorithm for convex polygons. Separation axis theorem can be explained intuitively with the analogy of shadow projected by objects. When a light source orbit around two non-colliding convex objects, it will project a shadow behind them. If the two objects are not colliding, after a complete orbit around the object, there will be a point where a gap is present between the two shadows, this is shown in Fig 10. When the two

objects are colliding there will be no position where the light can project a two separate shadow. This is show by Fig 11. Separating axis theorem states that the maximum number of side to project shadow from in-order to determine whether the objects are colliding or not is the same as the same as number of sides of the objects.

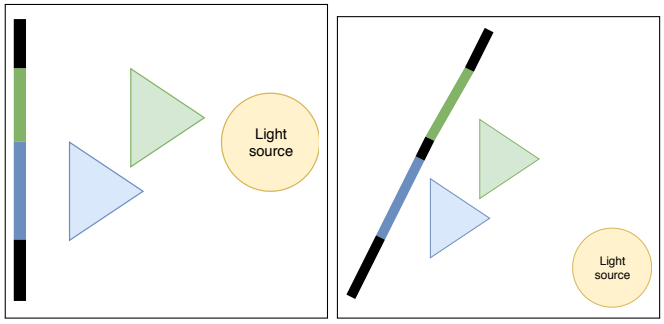


Figure 10: Diagram representing a light source projecting shadow behind non-colliding objects

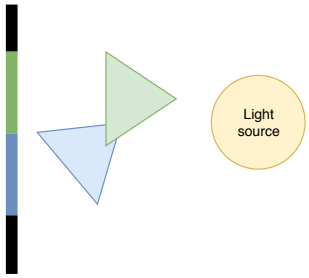


Figure 11: Diagram representing a light source projecting shadow behind colliding objects

```
def boundingBoxCollision(point, boundingBox):
    p1=boundingBox[:,0]
    p2=boundingBox[:,1]
    p4=boundingBox[:,3]
    p5=boundingBox[:,4]
    #lines to project point to
    arcsToProjectOn=[[p1,p2],[p1,p4],[p1,p5]]
    #projecting point to lines
    isColliding=True
    for arc in arcsToProjectOn:
        vectLine=arc[1]-arc[0]
        norm=np.linalg.norm(vectLine)
        pA=np.dot(vectLine,arc[0])/norm
        pB=np.dot(vectLine,arc[1])/norm
        pL=np.dot(vectLine,point)/norm
        if(pL>max([pA,pB]) or pL<min([pA,pB])):
            isColliding=False
    return isColliding
```

Method defined above implements the separating axis theorem. The method will only project the line to three lines since the method restrict it self to use with bounding box. A box only have three unique sides so the projection is only made onto three lines. Dot product of the screen vector and the point vector serves as equivalent of projection of shadow. If the dot product of the point and line is between dot product of the the two nodes then the point may or may not be colliding with the bounding box. However, if the projection of the point is greater or smaller than both of the nods projections then the two objects are not colliding. This concept is visually shown in Fig 12.

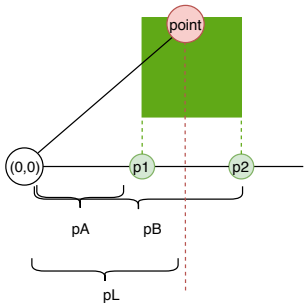


Figure 12: Separation axis theorem implemented to test collision between bounding box and a point

pA and pB is the projected point of the nodes. pB-pA is a side of the bounding box.

In this project, the boundingBoxCollision method is used to determine if the uniform field should apply to the particle or not and also to determine if the particle is colliding with tokens.

4.3 Runner kutter method

5 Possible Improvements

GUI lacked instruction and labels, the view position of three 2 dimensional figures should be stated in text. A new state machine automata was designed to allow for objects to change parameters and be deleted from the build world in build tab. Implementation of this would allow users to intuitively select and delete objects which would improve the interactivity of the GUI. This state machine automata is shown in Fig 13. Pickles can be used to save class instances, save and load system using the pickle package would save time on constructing the build world and training. This feature would also improve repeatability of experiments. Runner kutter method is accurate but computationally demanding method of simulation euler cromer method may have been more desirable for this application where simulation is repeated multiple times. Continuous fitness assignment algorithm

rather than a discrete system may have improved the level at which genetic algorithm could learn to optimize.

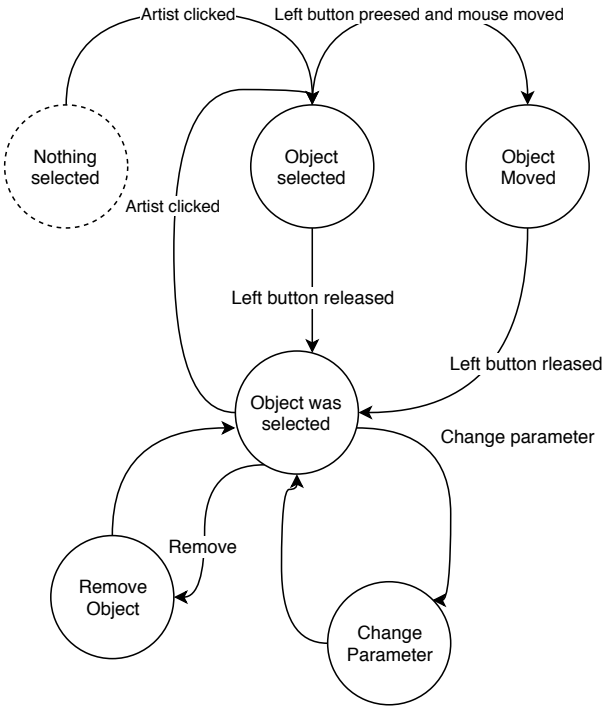


Figure 13: State machine automata that allow deletion and change of parameters of the objects

6 Post analysis of the project

Through out the project I noticed that my efficiency of progress was frequently hindered due to messiness state of the code. Through this experience I found three large factors that aid in maintaining a clean code.

1. Use obvious and intuitive identifiers and logic.
2. Function or method should only do one thing
3. Class relations should be formally planned before the coding of them are done.

In terms of the use of object oriented paradigm, areas in which I could have improved are stressing readable and settable attribute with property and setter decorators, use of decorators to make input validation efficiently and therefore improving the efficiency of debugging, modules should be individually testable rather than being tested after integration to the main program, use of user defined class when data structure is non trivial rather than a dictionary and use of aggregation when instance is shared across different classes. The time management of the project could have been improved by introducing pre-planned time schedule.

7 Summary

I successful programmed a genetic algorithm that optimize the problem of controlling a charged particle under specific path. I was also able to write an intuitive graphical user interface that allow users to build a simulation world with drag and drop procedure. I was able to observe some improvement being made as result of training of genetic algorithm. However, I could not observe a optimization that is better than how human would control the particle though this was not my aim.

8 Conclusion

I was able to apply genetic algorithm to control a charged particle with electric plates and direct them to a desired direction. Though genetic algorithm showed improvement on the way at which it control the charged particle, the result that I observed were far from the most optimal path and those problems that the algorithm was able to solve were trivial. Some of the contribution to this result may be due to lack of generations simulated but I believe the main cause of this result is the way in which the species are awarded with fitness in a discrete manner rather than continuous manner. This does not award the small positive change evolution and mutation make and will continuously kill off species that may have made positive change and will only award species that made a large enough change to be awarded by the fitness algorithm. These species that is awarded with the positive enforcement token may quickly be killed off in later generation since even if the change in specie is small, as soon as it leave the path of positive token, it will be awarded with 0 positive enforcement token. Therefore I advise those who want to attempt a similar project to use a continuous fitness determination algorithm rather than discrete scoring algorithm.

9 UML diagrams

I have included some UML diagrams which may help the readers to get to a specific section of the code efficiently. These UML diagram only show some important attributes and methods, less important ones are not shown. Variable types of a self-explanatory variables are also not stated.

9.1 Associations in the project

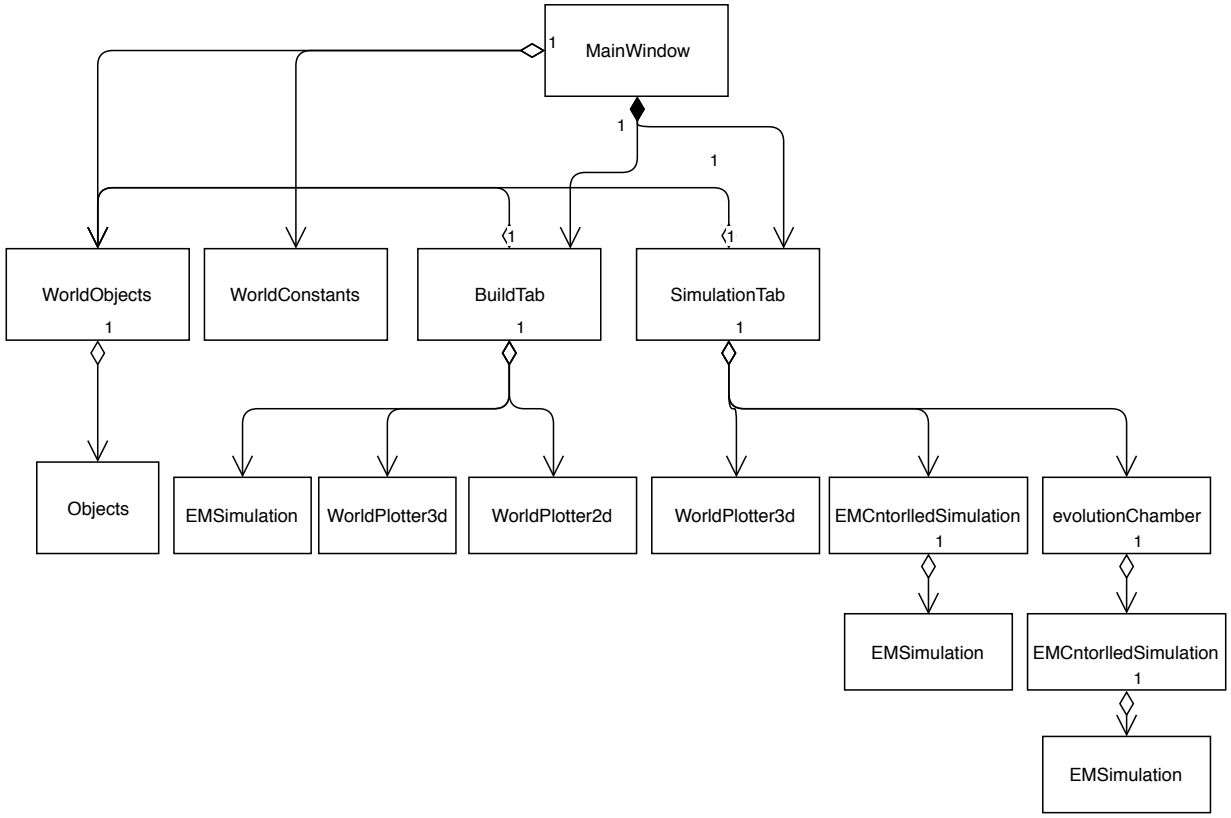


Figure 14: UML diagram that represents the association relations of the entire project

9.2 gui

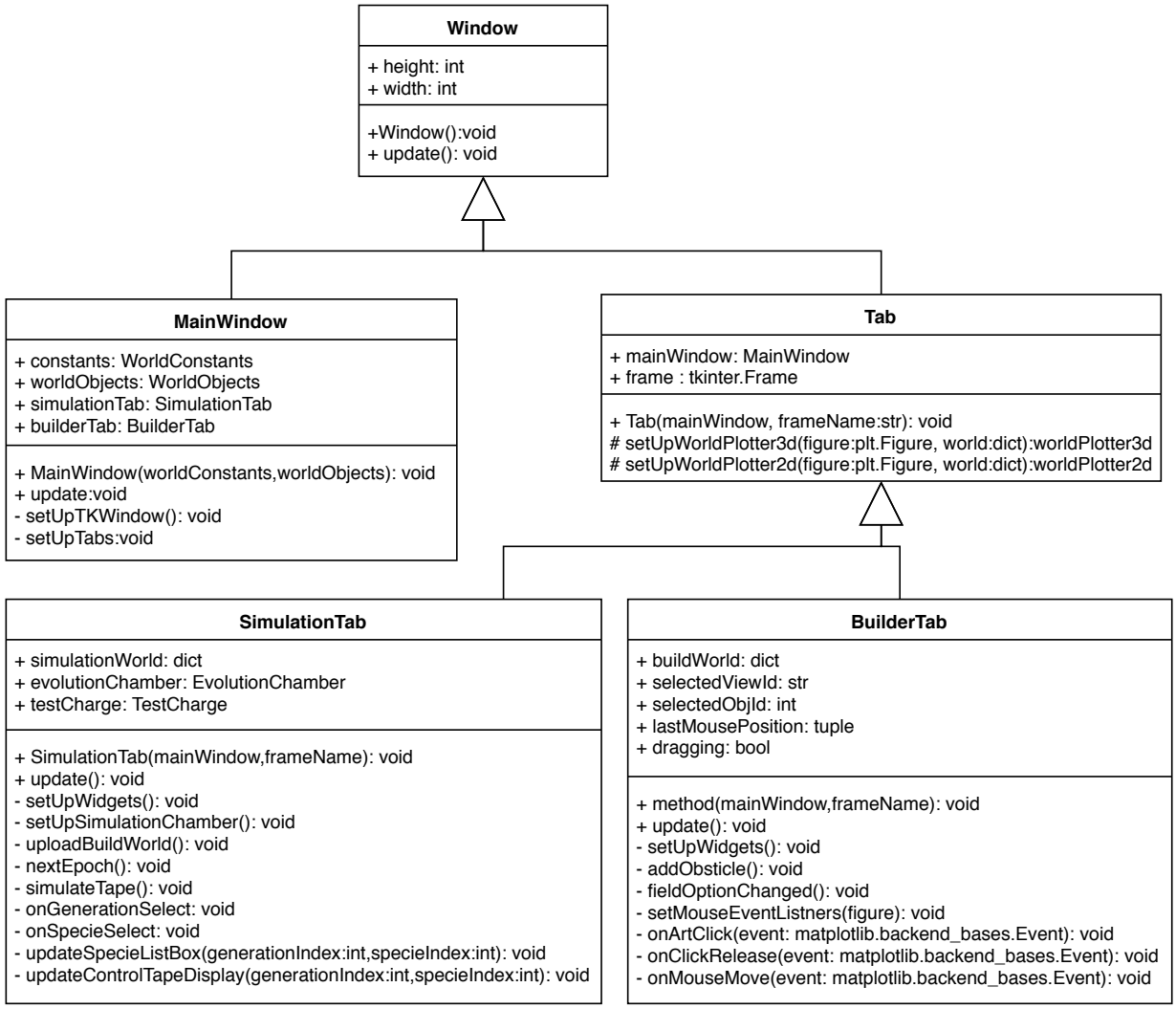


Figure 15: UML diagram showing class inheritance relation of gui

9.3 world plotter

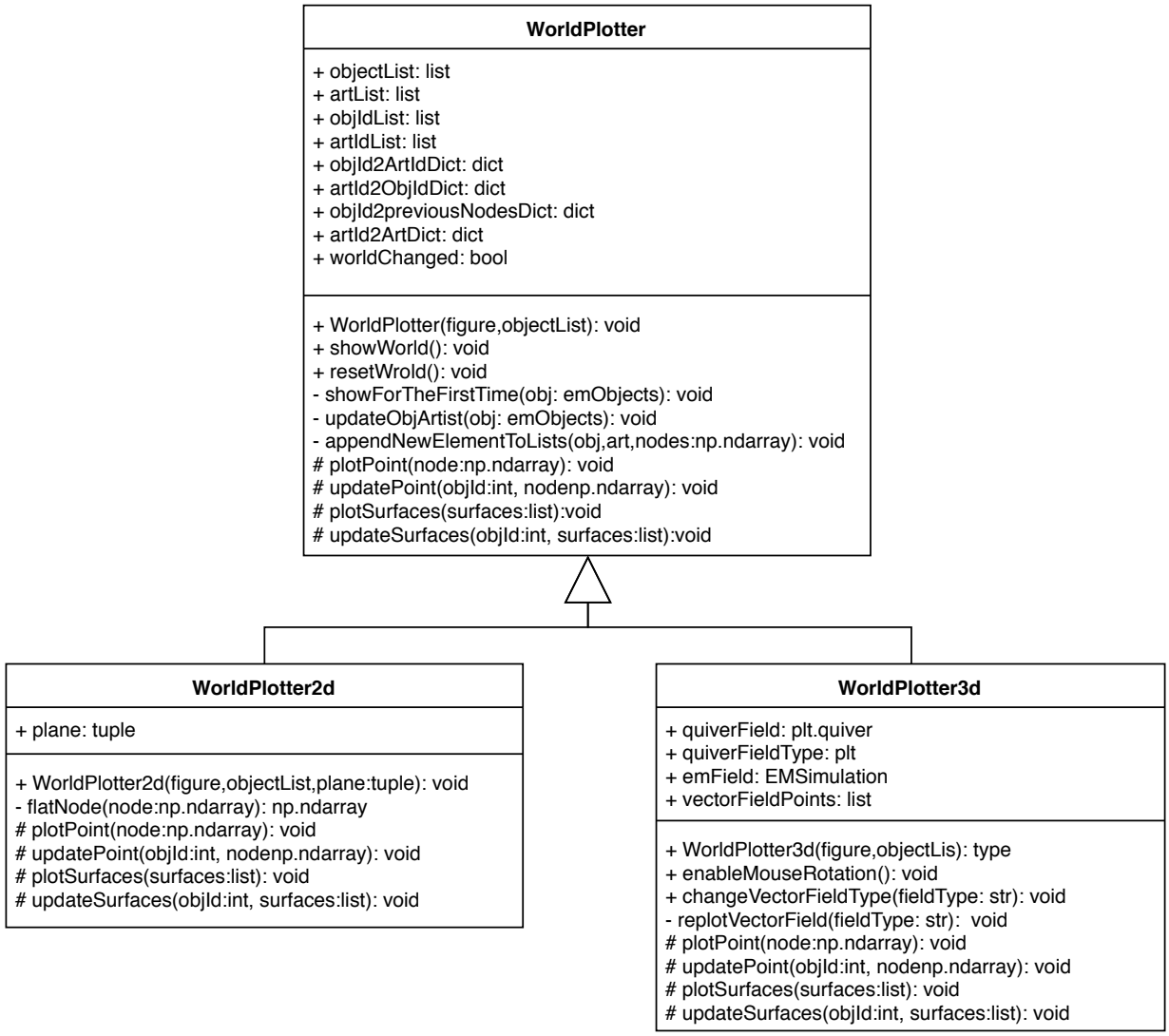


Figure 16: UML diagram showing class inheritance relation of world plotter

9.4 objects

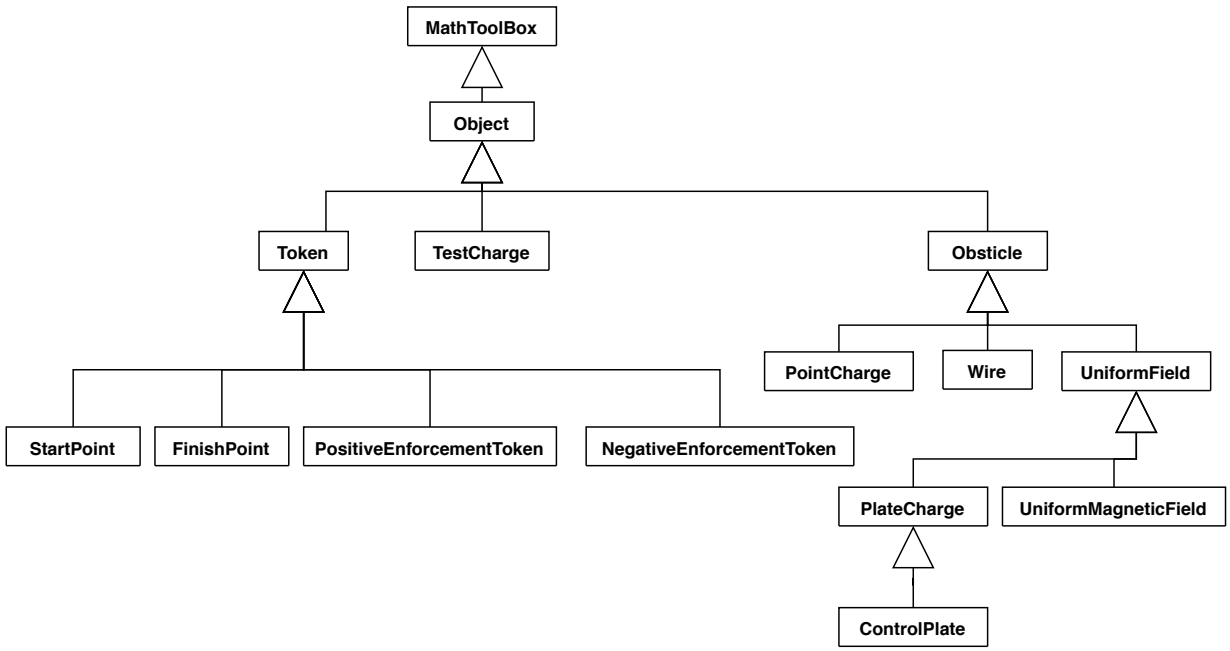


Figure 17: UML diagram showing class inheritance relation of world plotter

9.5 em simulator

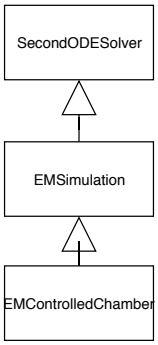


Figure 18: UML diagram showing class inheritance relation of em simulator