PLT Project Proposal

Thomas Rantasa (tr2286), ChengChen Sun (cs2890), and Benjamin Kornacki (blk2129)

1. Describe the language that you plan to implement.

A graphic description language which allows user to define a) 2D objects; b) movement of objects, and c) morphing of objects.

a) 2D objects: basic objects are defined, like circle and triangle. Parameters are passed when an object is created. Basic parameters will include position, size to locate an object within a plane. Additional parameters may include weight, color, etc. Users could define their own rules to change these parameters. Parameters can not only represent characters of an object, but also relations with other objects, like created from which object previously.

b) movement of objects: the object can move towards a direction in a predefined velocity in a certain period of time. Both direction and speed can be changed as time ends, but in this small period of time they keep unchanged. This can be used to simulate all sorts of physical movement. A minimum value of time slice must be defined and all valid time parameters must be multiple times of this value.

Also, object could circle around some axis at certain rotation velocity. With these two definition of velocity. Physical movements can be simulated.

c) morphing: given an object, morphing transforms it to another defined object, like transformation from circle to whatever object you like, in given period of time.

Based on these definition, our language is expected to simulate moving and morphing of all 2D objects. For example, Shifting, Rotation, Addition and Subtraction are four basic operations that can be introduced by users. Meanwhile, users can define their own constraints of objects to do whatever they want (for example, how objects interact with each other when their location satisfies some constraints). Examples are given in part 3.

2. Explain what problem your language can solve and how it should be used.

Our language will allow users to create relatively complex shapes and animations with very basic code and relative ease. Animation of shapes or even text can often be a very daunting task for a programmer. This language will allow a user to define most basic geometric shapes in a single line of code. More complex shapes can easily be created by merging multiple simpler shapes so the user does not need to have a background or understanding in graphics. Similarly, the language allows users to define various forces as well as multiple gravitational fields in one line. Given the very intuitive nature of the syntax, individuals with little background in programming can create very creative and complex shapes and movements with little (and easily understandable) code.

Since all shape design and movement will be based on a few number of physical principles, the user can combine the most basic of operations (for example move and gravity) in order to create multiple orbit fields without needing a single calculation or complicated data structure.

While this language will not offer 3-D graphic implementations, it should be used by users that fit the just described category. A user with little background in animation, or in need of simple yet powerful 2-D figure design and movement will be well served with this language. Given the right amount of effort, a user will be able to do significantly more with this language than described in this document, however these applications could often be implemented better or more effectively in another language.

3. Describe an interesting, representative program in your language.

Our language can be used for many purposes. The most basic purpose would be describing basic laws of physics such as kinetics and electric/magnetic fields. The user will have at his disposal the ability to make objects, define the location of the objects, move the objects, and create functions. With these basic tools (and the use of if statements and while loops) the user can create things such as velocity. In order to put an object in motion, the user will simply have to create a function that moves the object a certain distance with every tick of time (time will be a predefined function). If the user wants to create acceleration then, he simply will create a function that will change the velocity with every tick of time. The user can also define what happens if two object hit each other or the side of the screen. That is, the user can create can define functions that implement the following: if the location of a point on the border of object a is equal to the the location of a point on object b then have the two objects bounce off each other, or have the two objects stick together, or have the objects pass through each other. With these simple capabilities, the user can create simulations of basic physics experiments. The user can define: acceleration, force, momentum, impulse, inertia, circular motion, torque, work, power, and so much more. Once the user has these, he can create a basic simulation of the world as we know it.

Another fun thing the user can do is manipulate strings in a new way. The user will have the capability to treat each letter or each word as a a unique object. Therefore the user can move these objects as described in the previous paragraph. In the simplest case, the user would print “hello world!” and the string will appear in the middle of the screen, static and boring. A more fun and exciting example is to create unique gravitational centers for each character in “hello world!”. That is there will be one gravity center that only the character “h” will be drawn to, one point only “e” will be drawn to, etc. and one point that every character that is not in “hello world!” will be drawn to. Then the user places a random assortment of letters onto the screen in various locations (with lost of repeats, so it will take up the whole screen). When the program runs, all the letters will be drawn to there respective gravitational centers and when the dust settles all that will be left is “hello world!”.

Since this language can be used to describe motion in the real world and more, the possibilities are endless. The user can implement anything he sees, or anything he can imagine, so long as it is in 2D.

4. Give some examples of its syntax and an explanation of what it does.

We will have have basic integer and character variables that will be written in the standard way:

```
int x = 1;              \\this will create an integer variable named x with value 1
char y = ‘a’;           \\this will create a character variable named y with value a
```

The user will also be able to define object variables. There will be certain objects that are native to the language, such as circle and square. There will be implemented as follows:

```
object circle myCircle = {3, 4, 1};       \\this will create a circle named myCircle centered at
                                          \\location (3,4) with radius 1
object square myRectangle = {2,1,3, 5};       \\this will create a square named mySquare
                                              \\centered at location (2,1) with side lengths 3
                                              \\and 5
```

Movement of objects will be written as follows:

```
myCircle+;      \\this will move the center of myCircle to the right one position
myCirlce-;      \\this will move the center of myCircle to the left one position
myCircle++:     \\this will move the center of myCircle up one position
myCircle--;     \\this will move the center of myCircle down one position
```

Addition, subtraction, multiplication and division will be exactly as you would expect for integers. That is 1+2 = 3, 4-5 = -1, 4/2 = 2, etc. We can also use these operations on objects. For example.

```
myCirlce = myCirlce+1;              \\this will increase the radius of myCircle by 1
myCirlce = myCircle-2;              \\this will decrease the radius of myCircle by 2
myCirlce = myCircle*3;              \\this will triple the radius of myCircle
myCirlce = myCircle/2;              \\this will halve the radius of myCircle
```

If an object has two variables that describe it(i.e. a rectangle), then we can define operations that will adjust each dimension separately. For example:

```
object rectangle myRectangle = {0,0,1,2}     \\this will create a rectangle centered at (0,0), its
                                             \\horizontal dimension = 1, and vertical dim = 2
myRectangle = myRectangle .+ 1;                       \\now horiz dim = 2, vert dim = 2
myRectangle = myRectangle ,+ 5;                       \\now horiz dim = 2, vert dim = 7
myRectangle = myRectangle .* 2;                       \\now horiz dim = 4, vert dim = 7
etc.
```

If we simply use an operator for myRectangle without “.” or “,” (i.e myRectangle = myRectangle+1;) then this will add 1 to all side lengths.

We can also define addition for two objects:

```
object myObject = myRectangle + myCircle;        \\this will take the objects myCircle and
                                                 \\myRectangle and treat them as one object
```

```
                                          \\called myObject
myObject++;                               \\this will move the centers of myCircle and
                                          \\mySquare up 1
```

However, after myObject is created we can still access my rectangle and myCirlce independently. That is, in the previous example if we wrote, "myCircle+" this will move myCircle one place to the right and not myRectangle.

We can write velocity of myCircle as follows. We will create an integer variable called myCircleVeloctiy. The code will be as follows:

```
int myCircleVelocity = 3;

for (int i = 0; i < myCircleVelocity; i++) {
        myCircle--;
}
```

If we run this loop after every tick of time, we the myCircle will move down at a constant rate of 3 pixels/time.

Acceleration will simply be:

```
int myCircleAcceleration = 2;

myCircleVelocity = myCircleVelocity+myCircleAcceleration;
```

If we run this line of code and the velocity for loop after every time tick, myCircle will accelerate at a rate of 2 pixels/time^2.

The force of an object is simply:

```
int myCircleMass = 5;
int myCircleForce = myCircleMass*myCircleAcceleration;
```

Ans so on for all the principles of physics.