Lab 1 Design Document

1. Class Designs

1.1 Child Class

Description: The child class is my node class. This class is called and used whenever a new child is spawned. Each child represents a node in the linked list.

Constructor *Child(double x, double y)*:
My Child constructor takes two double parameters, which represents the initial coordinates of the Child object that was created when the "SPAWN" command was called. It will then initialize the *x_coor* and *y_coor* member variables with the (x,y) values. It will then set the **next_child* pointer to *nullptr* because the new node will not connect to the linked list yet.

Destructor ~Child():
When the destructor is called, it will set *x_coor* and *y_coor* to 0 and **next_child* will be *nullptr*. This ensures that object is properly finalized before its memory is deallocated.

Private Member Variables:

- ***double x_coor***: This variable stores the x coordinate of the child node.
- ***double y_coor***: This variable stores the y coordinate of the child node.
- ***Child *next_child***: This variable is a pointer which allows that instance of Child to point to another instance of Child as its "*next_child*", creating a chain of linked objects.

Member Functions:

1. *void set_next (Child *new_next_child)*
    - This is a setter function for the *next_child pointer. It will allow you to point the current Child object to another Child object given in the parameter, connecting the nodes in a linked list.
2. *double get_x_coor();*
    - This is a getter function for the x-coordinate of the Child object and provides access to the private *x_coor* variable.
3. *double get_y_coor();*
    - This is a getter function for the y-coordinate of the Child object and provides access to the private *y_coor* variable.
4. void *set_x_coor (double x)*
    - This is a setter function for the x coordinate, which allows you to update the x-coordinate
5. *void set_y_coor (double y)*
    - This is a setter function for the y coordinate, which allows you to update the y-coordinate
6. *Child *get_next()*
    - This is a getter function for **next_child* and provides access to the private **next_child* pointer.

2.2 Game Class

Description: My Game class is my linked list class. This class is called and used whenever Child objects need to be added/deleted from the list or the linked list needs to be modified depending on various commands and inputs.

Constructor *Game():*
My constructor initializes the *head_child* pointer to *nullptr* when a new Game object is created.

Destructor *~Game():*
My destructor iterates through the linked list, deletes each Child object and set their pointers to *nullptr.* This will free up the memory used by the list and prevent any memory leaks.

Private Member Variables:

- *Child *head_child:* This is a pointer variable representing the beginning of the linked list.

Member Functions:

1. *Void spawnNewChild (double x, double y)*
    - This function will check if the (x,y) co-ordinates provided in the parameters are in the first quadrant, then it will create a new object and add it to the list.
2. *Void removeChild (double x, double y)*
    - This function will search in the list for a Child object that has the same (x,y) co-ordinates as given in the parameters and remove that object from the list.
3. *Void moveChildren (double t)*
    - This function iterates through all the Child objects in the list and moves them based on the given *t* parameter. It will calculate their new (x,y) co-ordinates and updates each child
4. Void numOfChildren()
    - This function iterates through the list and counts the number of children in the list
5. *Void printChildren (double D)*
    - This function will iterate through the list and print the co-ordinates of the Child objects whose distance from origin is less than the given *D* parameter.
6. *Void over()*
    - This function determines the outcome of the game
7. Child *get_head_child()
    - This is a getter function and provides access to the *head_child* pointer

2. Expected Runtime

2.1 "TIME" and "PRT" commands

When the "TIME" and "PRT" command is called, it will call the *moveChildren* or *printChildren* function respectfully. Both functions have a single loop that will iterate over the Child objects to look at all the children in the list. Thus, the loop's runtime will depend on the number of children (n), so the expected runtime will be O(n).

2.2 "SPAWN", "NUM", "OVER" commands

When these commands are called, it will call the *spawnNewChild*, *numofChildren*, or *over* function respectfully. These functions perform constant simple operations that do not depend on the number of children. Thus, the runtime will be constant and will be O(1).

3. UML Design