Rohan Venapusala
83938087
Assignment 6 - Midterm

Part 1
https://github.com/ronvenna/inf123
Part 2A

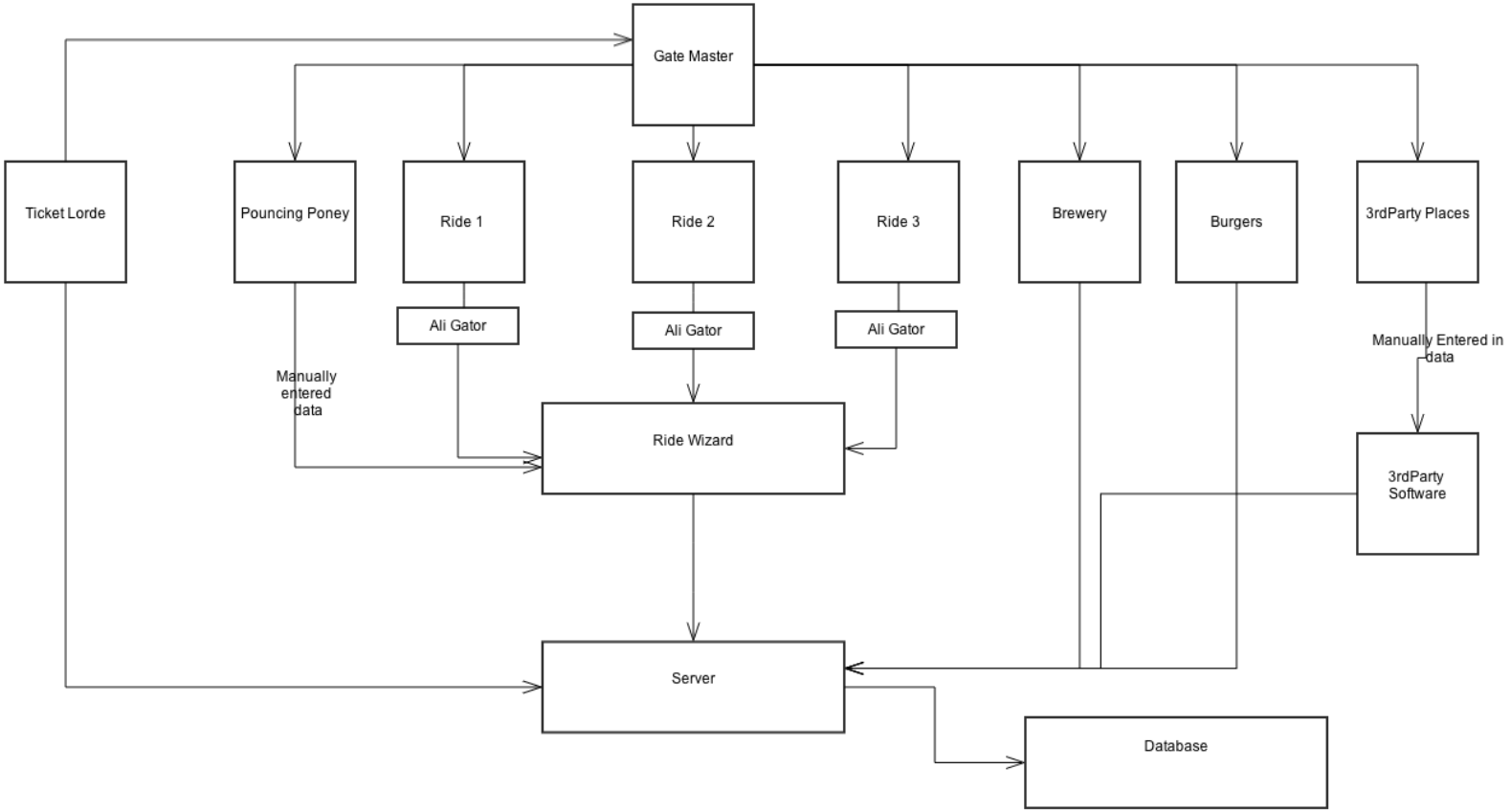
Peer to Peer
Pros
Each element of the system could act as “nodes” which will all be interconnected and share resources amongst each other. This makes sure each resource is efficiently used and takes away being redundant. This also could save a lot of computing resources and power since all the systems are connected with one another which intern could save a lot of money in the long run for the park. This also saves money for expensive centralized servers since each individual systems are used to access the data. A peer to peer network is also much easier to set then other networks such as a client-server.

Cons
Since all the individual systems will be sharing resources, they are both the suppliers and the consumers of the resources, this may cause problems in data manipulation such as when one system is trying to manipulate the data another system is doing the same which could cause the right data to be overwritten. An example of this is someone scans their id entering a restaurant and the immediately goes on a roller coaster, if the Ali Gator and the restaurant system are both trying to manipulate the data at the same time, this may cause deadlock. Also there is a security flaw with this type of network because if one system gets compromised this could cause the whole p2p to get infected.

Client-Server
Pros
The advantages of the client-server model is that all the data is housed in a central location (the database) and each system can access the data one at a time, which fixes the problem of the same data being manipulated by two systems at once. This also fixes the problem of security as backing up of the data and security is controlled centrally. This is also a straight forward setup of the network and lots of documentation and templates for this model exits open source.

Cons
The biggest con of this system is the cost. The server will be expensive to purchase and must be maintained and upgraded when needed. Scaling the server long-term might cause problems as well. Also in a client server system, if any component of the network fails the whole centralized system will go down, which may cause the whole theme park to go down.

Q2. Provide a high-level architecture of the whole system. You can justify your decisions in a small accompanying text (less than one page).

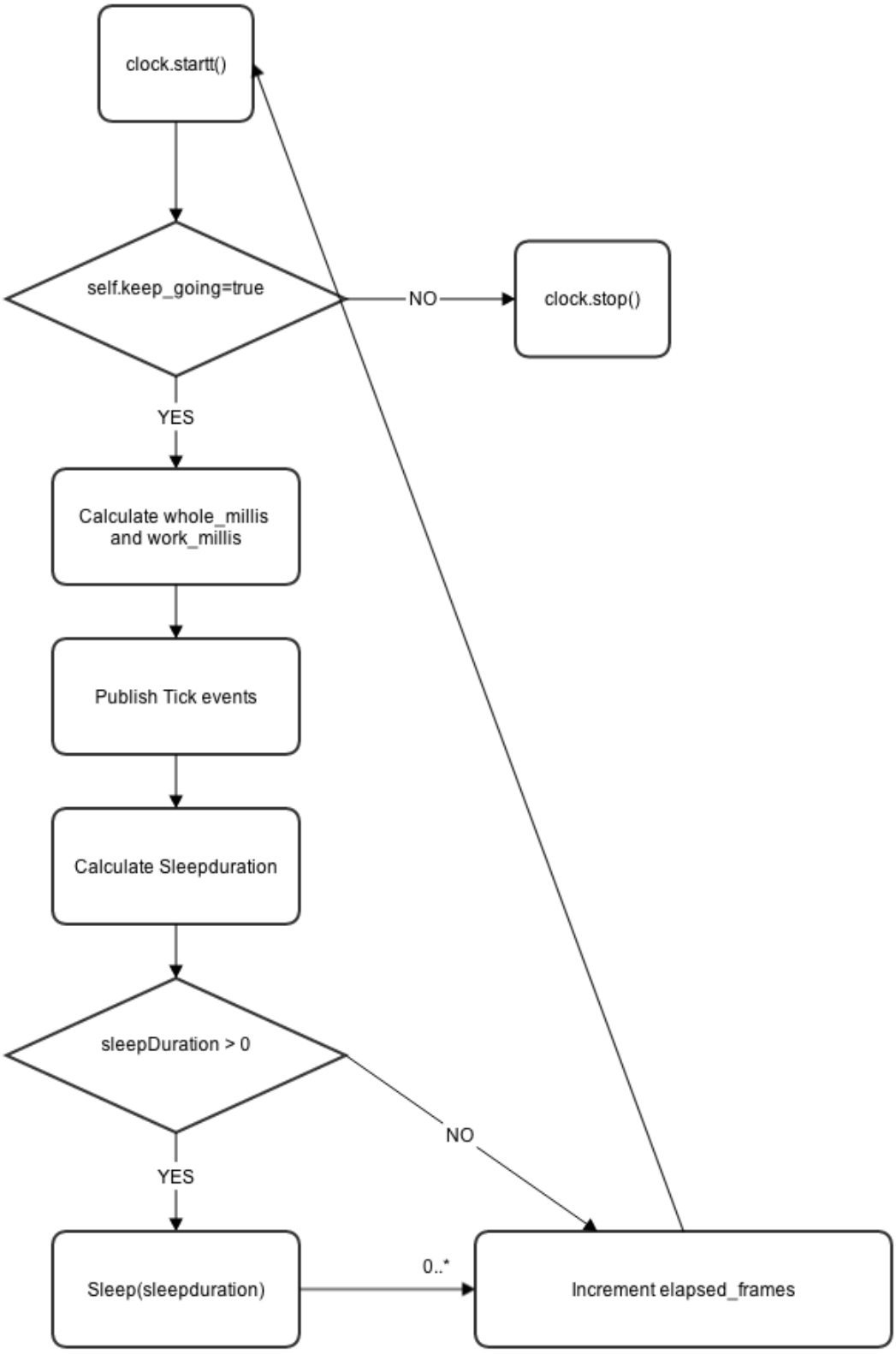

Gate Master
Ticket Lorde
Pouncing Poney
Ride 1
Ride 2
Ride 3
Brewery
Burgers
3rdParty Places
Ali Gator
Ali Gator
Ali Gator
Manually entered data
Manually Entered in data
Ride Wizard
3rdParty Software
Server
Database

Client Server Model

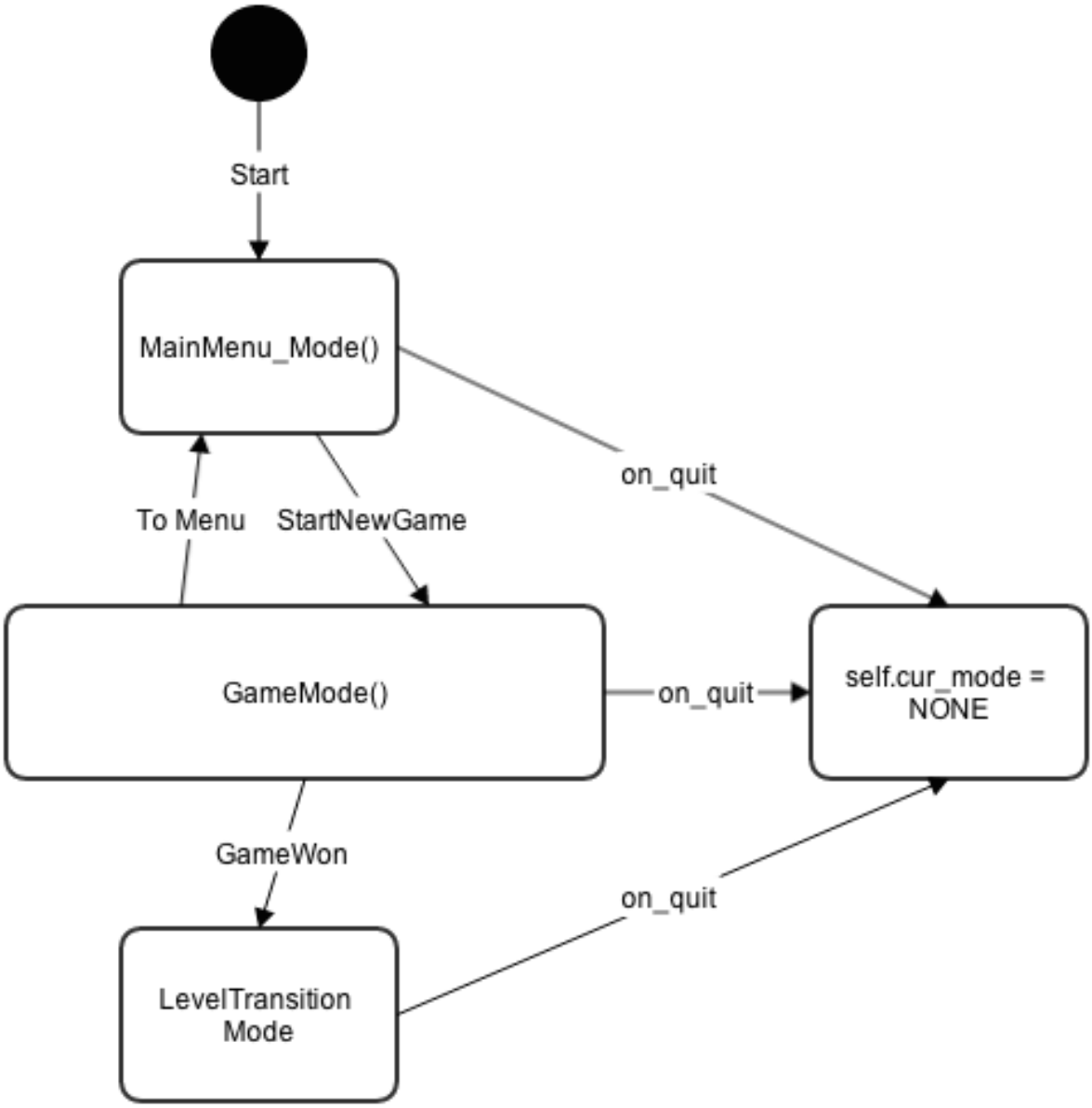
I went for the client server model mostly because of security. A multi million dollar amusement park needs to be as secure as possible and I feel like the client server model will be able to provide that over the peer to peer mode. Also the revenue the amusement park will be making should be enough to cover the costs of the network. The pros outweighed the cons in the Client Server model over the peer 2 peer model. In my client server model, the tickets are bought by the ticker lord system in batches of 1000 the gate master system will than scan those tickets. The gate master will also be taking care of the rewards when the user exits the park. once the ticket is scanned an ID is given to the user. When the user goes on rides the ID will be processed and sent to the central ride server system which then dispatched in the correct format to the server and manipulates the data on the database. For the pouncing pony ride that will be added later the data must be manually entered and this get sent directly to the server. For the food, each restaurant will send their data in the correct format directly to the server. For third part restaurant this needs to be done manually, and the 3rd party software will send that data to the server. Whenever anything goes to the server, the server manipulates the data on the centralized database. This architecture is very straightforward, however some hardware must be upgraded to scale the amount of people coming in.

2B. Reverse-engineering

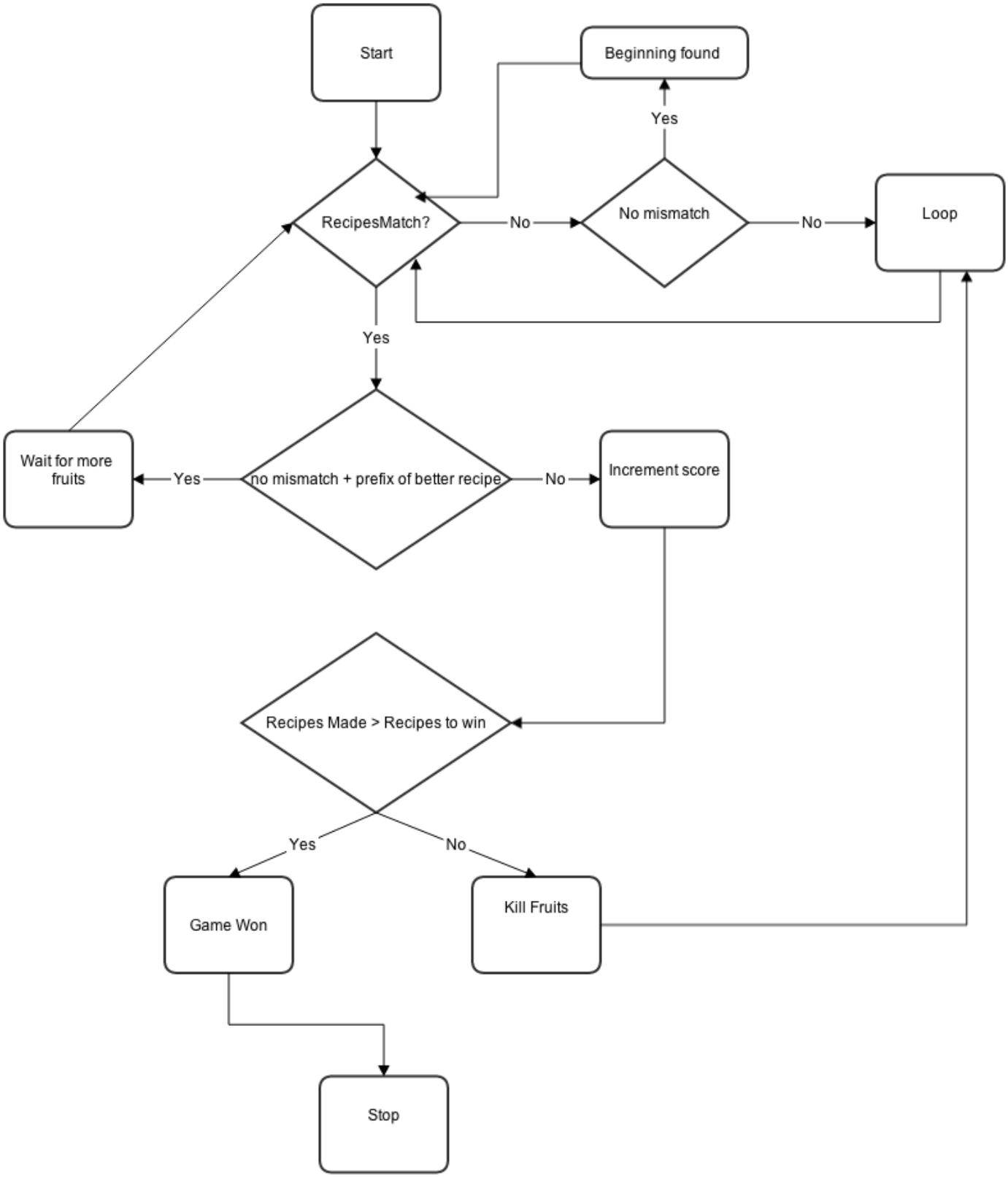
Q1. Make a flow chart of the clock. There should be less than 10 boxes in the flowchart. (15 points)


clock.startt()
self.keep_going=true
NO
clock.stop()
YES
Calculate whole_millis and work_millis
Publish Tick events
Calculate Sleepduration
sleepDuration > 0
NO
YES
Sleep(sleepduration)
0..*
Increment elapsed_frames

Q2. Make a state machine diagram explaining the high-level functioning and structure of the game modes. The edges should be the modes and the vertices the events triggering mode changes. (15 points)


Start
MainMenu_Mode()
on_quit
To Menu
StartNewGame
GameMode()
on_quit
self.cur_mode =
NONE
GameWon
on_quit
LevelTransition
Mode

Q3. Make a flowchart of the game logic. The chart should have a start state, an end state, at least one diamond checking for game over, at least one diamond checking for game won, and at least one diamond checking for recipes. Do not forget the score. (20 points)


Start
Beginning found
Yes
RecipesMatch?
No
No mismatch
No
Loop
Yes
Wait for more fruits
Yes
no mismatch + prefix of better recipe
No
Increment score
Recipes Made > Recipes to win
Yes
No
Game Won
Kill Fruits
Stop