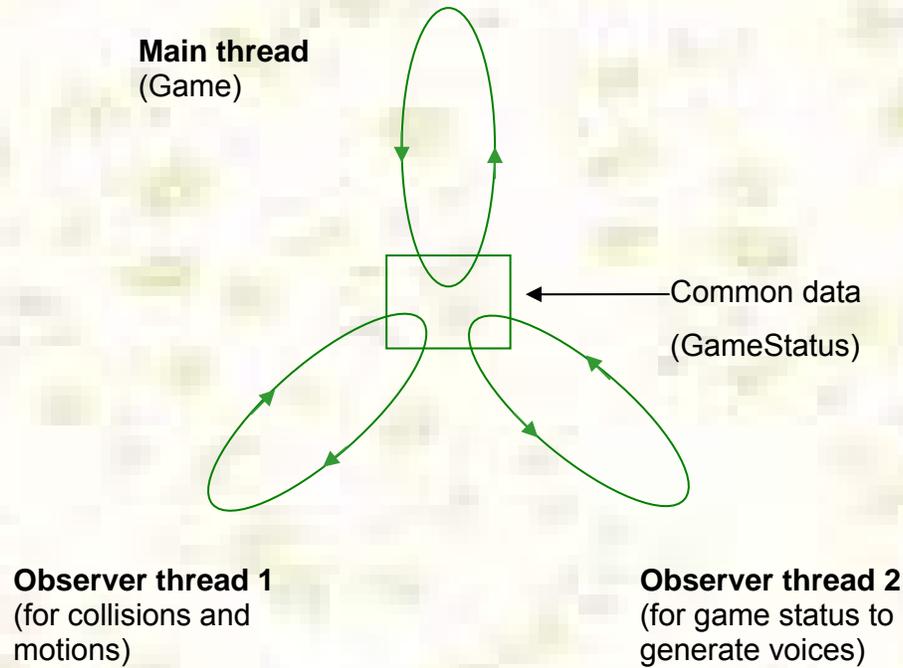


Introduction

Carrom Game is a C++ multithreaded application with OpenGL and OpenAL for sounds and graphics. This document explains few concepts used in this program so that anyone interested in concepts probably can try these out.

Basic architecture



There are three synchronized threads in this program. The main thread is the game itself and the second thread is to observe collisions and motions of coins and generate sounds.

The third thread simulates friends around the user (human player) and it observes game events / user actions just like friends watching, and generates conversations probabilistically for those events detected.

Basic flow of the Program

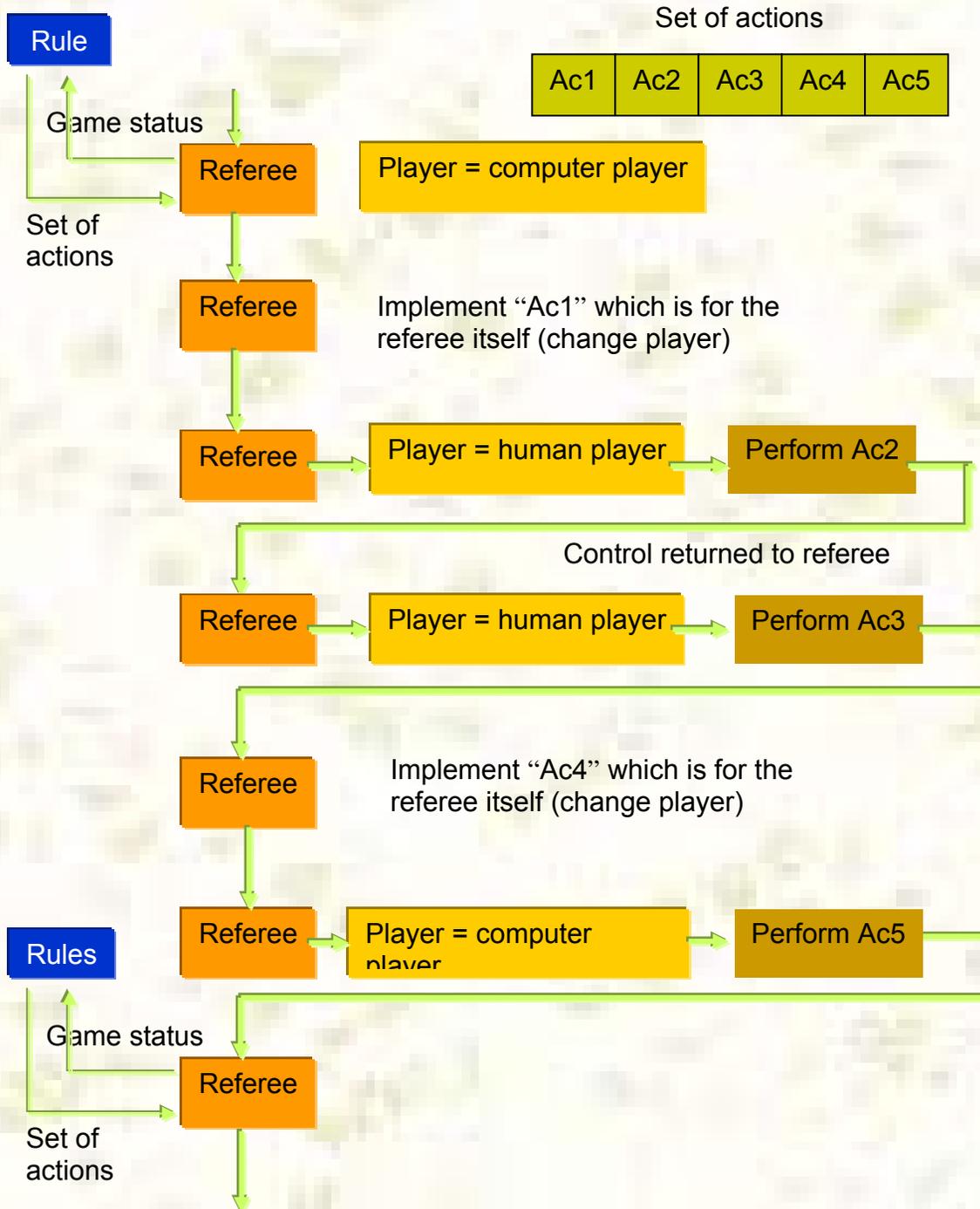
Game progresses as a sequence of polymorphic commands issued by the "Referee" object to the present "Shooter". "Referee" updates the "GameStatus" and sends it to the "GameRules" object. "GameRules" returns a set of actions to be carried out for the particular situation.

Referee carries out the action by itself or it issues a command to the present shooter to carryout the action. The player executes the action and returns control back to the Referee.

This continues until all actions are implemented. Once all actions for the particular rule are carried out Referee updates the game status and sends it to rules and rules returns another set of actions. This process starts when the game starts and continues until game ends.

Example scenario – computer player had just pocketed the striker along with its own side coin and returned control to the referee. (The rule is to take out the coin along with a penalty coin but retain the strike with the same player. The coins are taken out and placed by the opponent player).

P.T.O



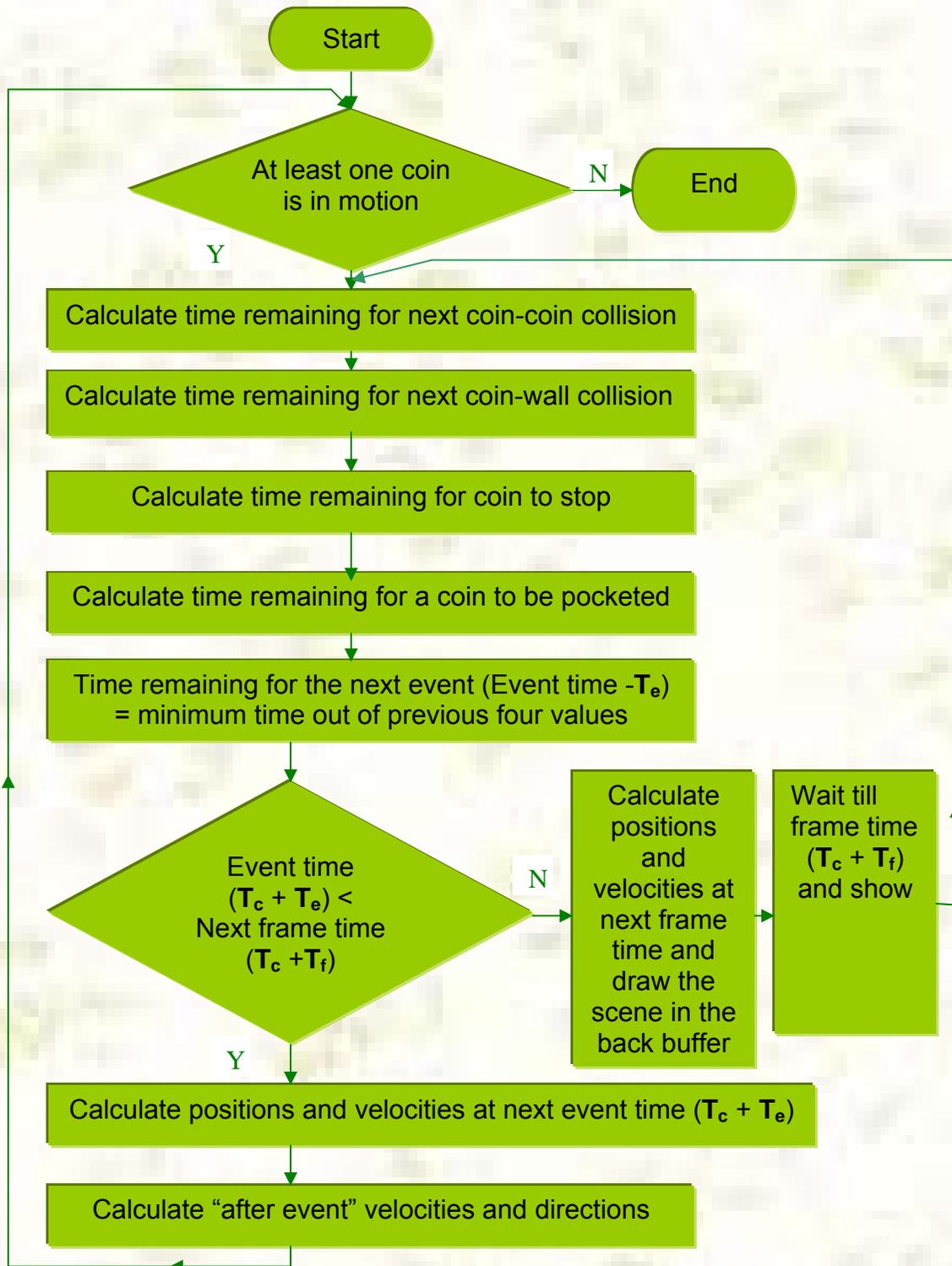
Here Ac1, Ac2, Ac3, Ac4 and Ac5 are change player, command player to pick a coin out, command player to pick a coin out, change player and command player to play respectively.

Motion algorithm with collision prediction

Following collision prediction method was developed without having any intension to create a game. Therefore OpenGL provided collision detection is not used in this program. The Initial intension was to create a mechanism to detect collisions of small molecules with high velocities. This method works accurately virtually for any size and any velocity provided that relative accelerations are sufficiently small. All other motion related events such as coin being pocketed, stopped and collisions with walls also are predicted.

Generally in collision detection methods, collisions are detected per each frame. This is done by checking whether two or more objects are overlapped in a particular instance of time. However this method might fail if objects move very fast and the sizes of objects are relatively small. This is due to the fact that collisions might occur in between two instances of calculations.

The basic flow chart and explanations are below.

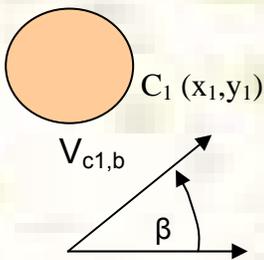


a) Calculating time remaining for next coin-coin collision

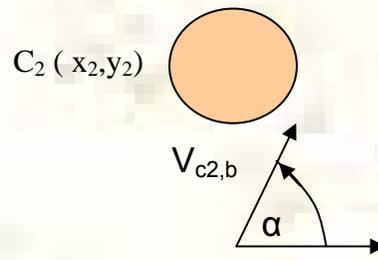
For this, relative velocities and relative paths of coins are used. Collision location in the relative path is determined and time taken for the coin to travel to the collision location is calculated. This calculation is done with all combinations of coins and the minimum time is considered as the time remaining for next coin-coin calculation. For a small acceleration and for a small time period it can be assumed that velocity change within the time period is negligibly small.

1) calculating relative velocities and relative paths

Consider two coins each having velocities and directions as depicted in the following diagram.

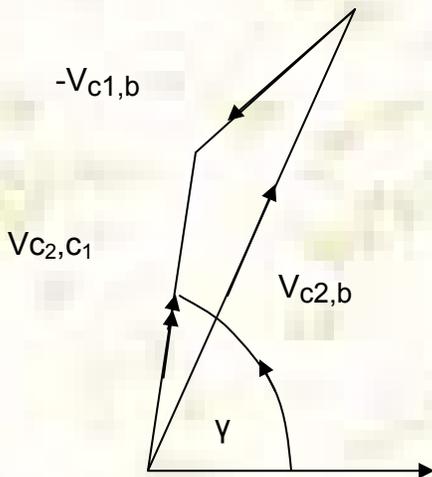


Relative velocity of coin1 at time t



Relative velocity of coin2 at time t

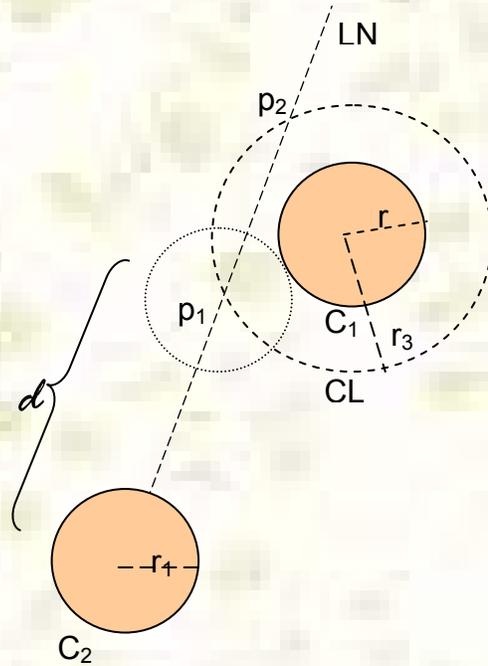
Since, $V_{c2,c1} = V_{c2,b} + V_{b,c1}$



Velocity of C_2 relative to C_1 at time t

Therefore the relative path is $y = mx + c$ which satisfies (x_2, y_2) and whose $m = \tan(\gamma)$

2) determining point of collision and time taken for it



Here,

r_1 -- radius of C_1

r_2 -- radius of C_2

r_3 -- $r_1 + r_2$

LN -- $y = mx + c$

CL -- $(x-x_1)^2 + (y-y_1)^2 = r_3^2$ Circle whose center lies on x_1, y_1 and radius r_3

p_1, p_2 -- intersecting points of LN and CL

d -- distance C_2 travels on relative path before colliding with C_1

Finding p_1 and p_2 is done by solving equations LN and CL .

From points p_1 and p_2 , the one on which coins collide is the one closer to C_2 .

i.e

$$\text{If, } (x_1 - x_{p_2})^2 + (y_1 - y_{p_2})^2 > (x_1 - x_{p_1})^2 + (y_1 - y_{p_1})^2$$

The required point is p_1 , else p_2 .

Whether this is a past occurrence or not is determined by using coin location on the relative path after very small period of time. If the new point is closer to the collision location it is a collision in the future. In this way the collision is predicted.

$s = ut + \frac{1}{2}ft^2$ equation is used to determine the time taken for the collision by substituting relative velocity of C_2 for “ u ” and deceleration component due to friction between the coin and the board for “ f ”. In this way the collision time is predicted.

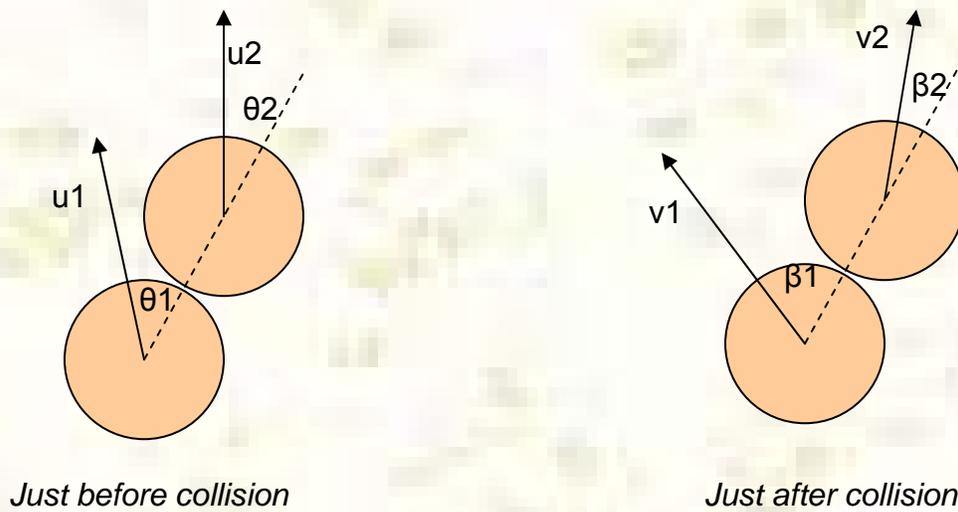
This whole calculation is done for all pairs of coins and the least time is taken as the minimum time for next coin-coin collision.

b) Calculating the time remaining for the next coin-wall collision

This is done without considering relative velocities. Since the distance between the edge (wall) of the board and the coin is known the time taken for collision is determined by using $s=ut+\frac{1}{2}ft^2$ by substituting distance for “ s ”, deceleration for “ f ” and velocity for “ u ”.

This calculation is done for all the coins that are in motion and the least time is taken as the minimum time for next coin-wall collision.

c) Coin- coin collision calculations



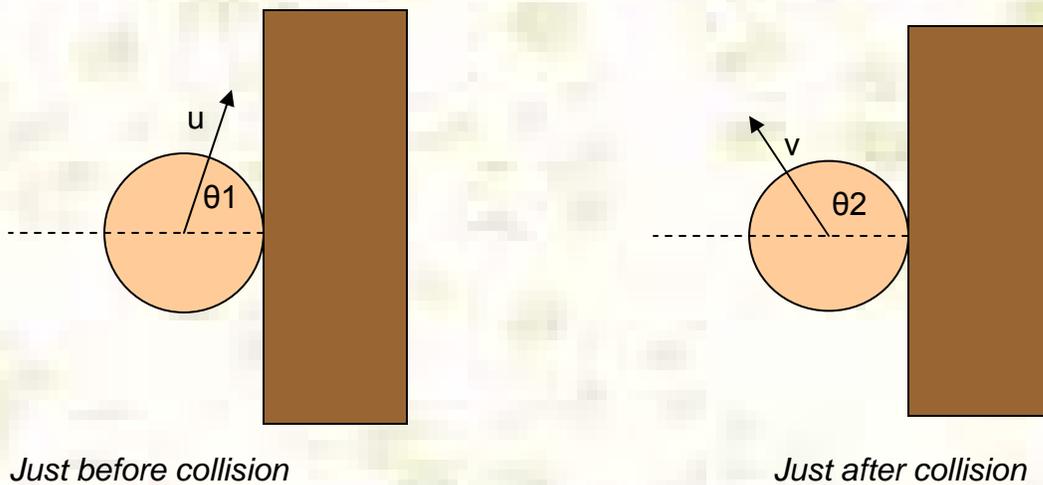
As we know, relationship between velocities in the direction of collision is,

$$v_1 - v_2 = -e(u_1 - u_2)$$

The velocities in a direction normal to collision direction are unaffected.

By using this and other properties of motion, the new velocities and directions are found.

d) Coin-wall collision calculations



Collision calculations with walls are done in the same manner. Velocities in collision direction obey the above Newton's formula and the velocities normal to it won't change.

Next position calculation with equations of motion

This is done by using $s = ut + \frac{1}{2}at^2$ equation by substituting X and Y directional components of frictional force due to friction between the coin and the board, initial velocity of the coin and time the coin had been traveling.

Computer Player

Computer player works like this. First it analyses the situation and applies a set of rules to determine the best shot.

Computer player's analytical functions include the following.

- Categorizing coins to a set of categories based on coin location
- Determining all possible shots for each an every coin (direct, back etc.)
- Determining clusters of coins
- Determining possible shots for all clusters with own side coins
- Determining possibilities of shooting a coin to a more convenient location

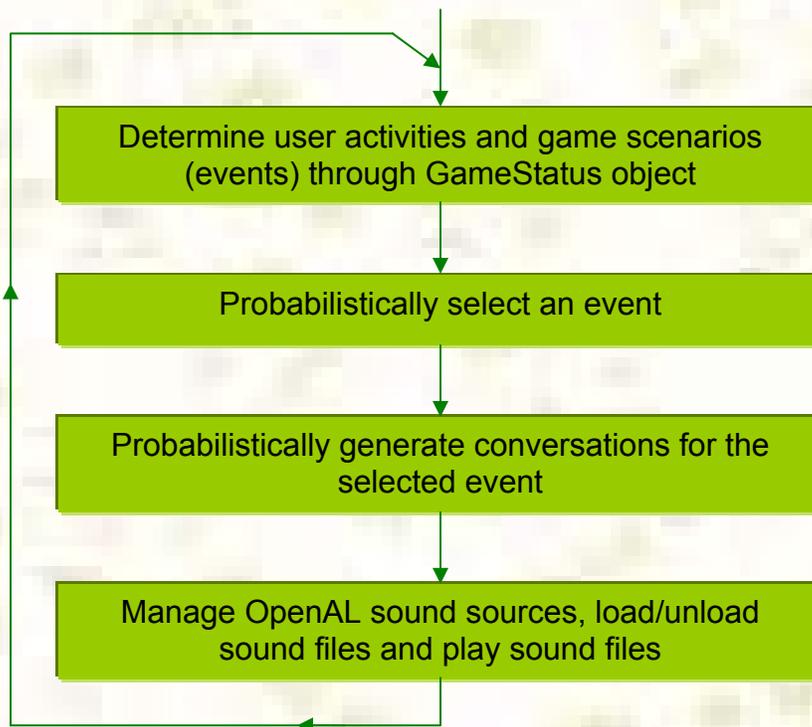
Then it uses a set of rules to select the best shot with the findings of analytical functions. Currently these rules are just test rules that I used to test analytical functions. However it seemed even those simple rules are quite enough for the game.

Thread 1 - Collision and motion observer thread

Function of this background thread is to monitor collisions and motions of coins, manage OpenAL "Sound Sources" and play sound clips as appropriate. It is quite possible to include the functionality of generating collision and motion sound effects to the main thread it self. However this functionality was separated to make sure collision calculations are done in real time, without any other burden.

Thread 2 - Game events and user activity observer thread

Function of this thread is to simulate friends around the human player. Following diagram illustrates basic flow of the thread.



a) Detecting user activities and game scenarios

Basic data used to determine user activities and game scenarios are read from "GameStatus" structure which is updated and read by all three threads.

Thread uses a set of analytical functions to determine user actions and game scenarios. Some analytical functions use only data from GameStatus object and other functions use ActionLog object which is updated by other analytical functions.

When a particular analytical function determines an action of the user it is stored in "ActionLog".

ActionID	Time
A _i	T _i

Action

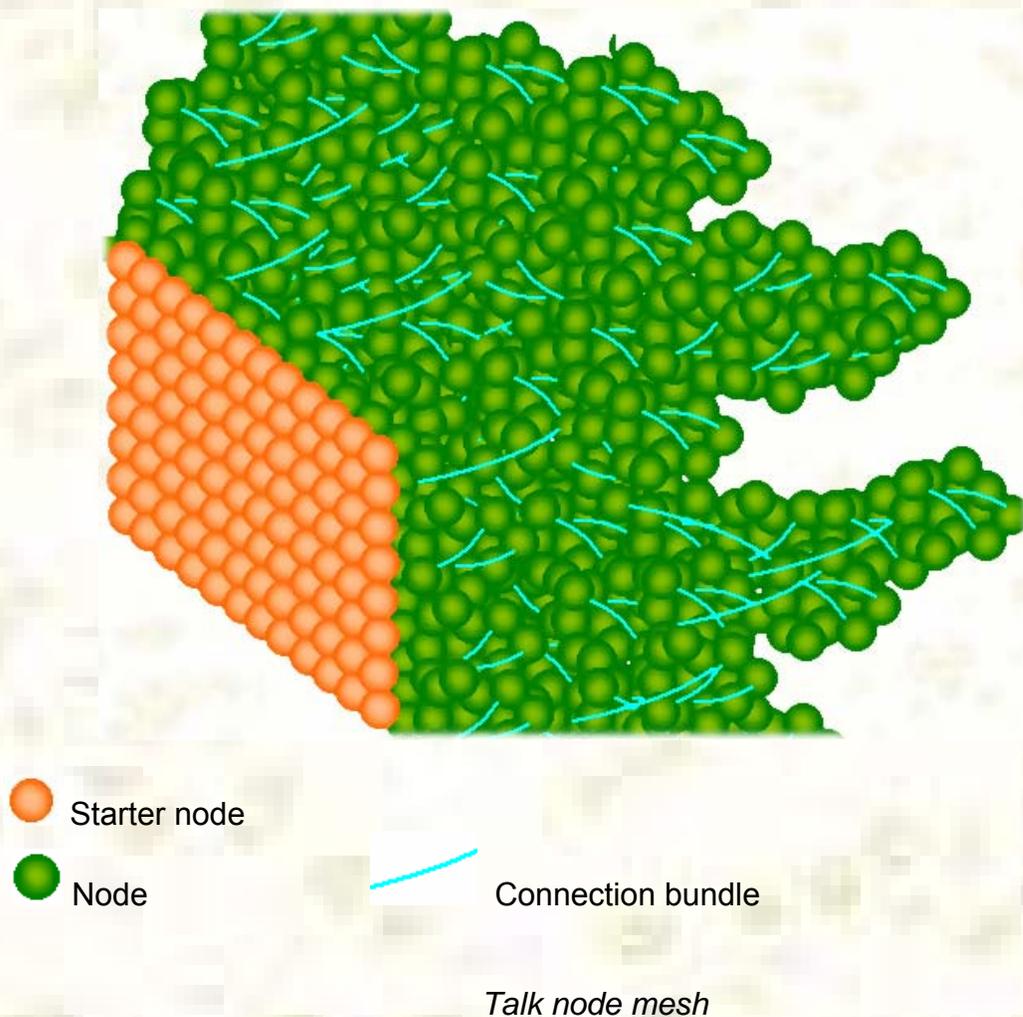
A1	A2	A3	A4	A5
----	----	----	----	----

ActionLog (An array of "Actions")

Once these events are determined one of the events is selected probabilistically. Once selected, probability of occurrence of that particular event is reduced by a value unique to the event. However the probability of occurrence of the event is gradually incremented with time until it reaches its original value.

b) Generating conversations for the selected event

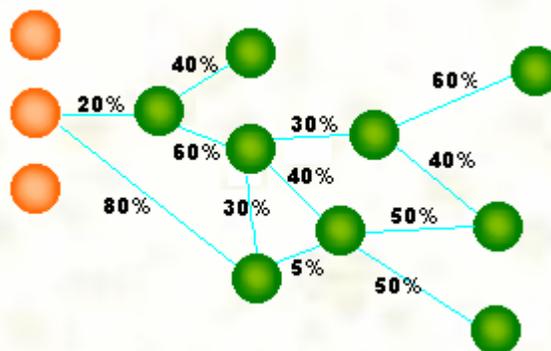
This is done by using an interconnected mesh of nodes as depicted in the below picture.



- “Talk node mesh” is a mesh of interconnected of nodes.
- Each node represents a voice file with necessary information to play it.
- Starter nodes can start conversations for one or more events
- Nodes are connected with zero or more “connection bundles”.
- A connection bundle may contain one or more connections.
- Connection bundle connects two nodes.
- Connection represents connection between two talks
- Connections are one way, a second talk after a first talk, etc.
- Connection has properties such as probability of occurrence, relative delay to start playing etc.

Conversations are generated and played in the following way

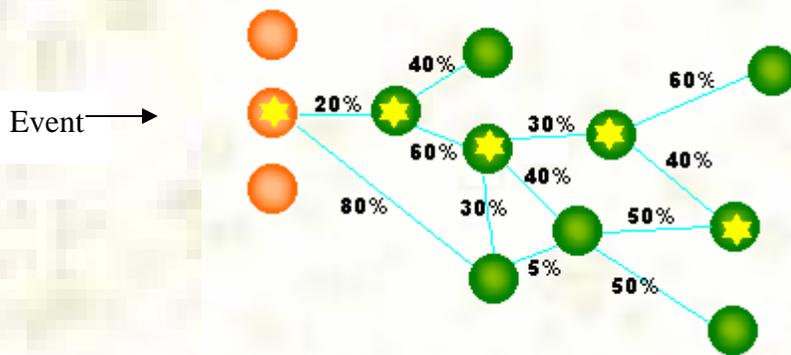
- 1) Event occurs.
- 2) One of the starters (probabilistically) for the particular event gets triggered.
- 3) One of the connections to a next node gets triggered (probabilistically).
- 4) This triggering process goes until it finds the end of nodes or stopped probabilistically.
- 5) In this way a conversation is logically generated.
- 6) Information required for playing sound files are taken from triggered nodes.
- 7) Sound files are opened, buffered and played with appropriate delay in between them and buffers are released.
- 8) Once a particular sound file is played, “probability of occurrence” in the respective connection is decremented by a specific value unique to that connection. (This probability of occurrence is then gradually incremented with time until it gains original value).



Sample Node mesh

Above diagram shows an example mesh with probabilities of occurrence for an event e_i .

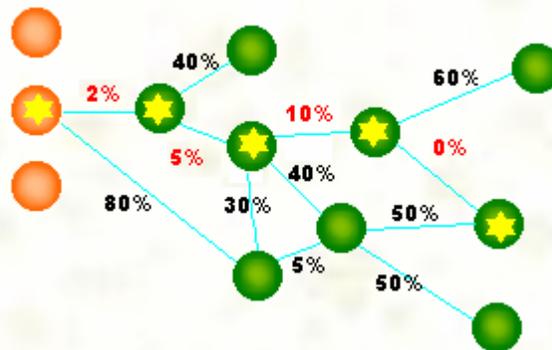
Let's assume the triggering mechanism triggers above mesh as below.



Sample node mesh with triggered nodes

Once the triggering process is complete, sound file name, relative delay and node id are taken from these triggered nodes and sent to a sound player object which plays these files one after the other with appropriate delay between clips by using relative delay values.

The “node mesh” is informed just after a file started to play. Then the node mesh will decrement the probability of occurrence for the particular connection by a value unique to the connection.



Sample node mesh after all files are played

(These values are then gradually incremented by values unique to each connection until they gain their original values)