CS 154 Project Report

Akshat Chugh	Kushagra Juneja	Kritin Garg
170050019	170050041	170050028

Carrom King

Introduction :-

This is a game based on the popular board game Carrom. We have implemented both single-player and two-player game modes.

Overall Idea of design :-

We have used a global state variable to keep track of all the entities on the carrom board. The coins and striker are represented by a struct which encapsulated all their properties.

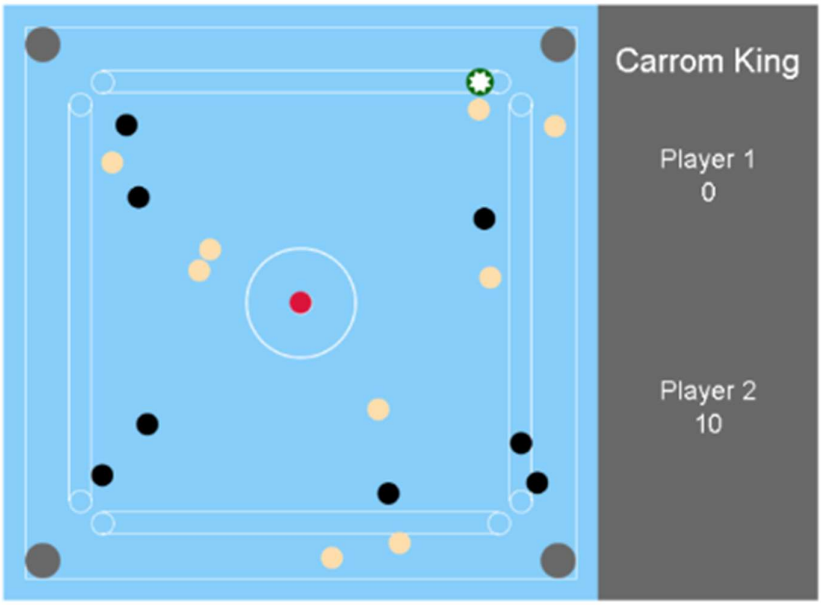
We have used principles of collision , momentum and force heavily as well as a lot of co-ordinate geometry.

Other notable ideas used-

- **Macros :-** Use of macros for creation of for loop .
- Implementation of nested for loops using the for loop macro.
- **Exception Handling** :- We have used this multiple times to handle errors such as division be zero.
- **Graphic Libraries** :- Use of 2htdp/image and 2htdp/universe , use of all the mouse-events for perfect positioning and movement of striker. To-draw, On-mouse, On-tick all three classes have been used in the implementation.
- **Abstractions :-**
- All the coins and strikers are reduced to a single structure.
- Use of nested maps to make the code more elegant.

- All the structures have been defined mutable so as to save memory .
- **AI Algorithm** :- Our AI considers every coin and two holes in front of it and calculates whether it is possible to sink a coin into a hole . It does so by explicitly calculating the angle required to sink a coin . Special priority in order queen to white to black coin .

Input and output Examples :-



Limitations and bugs :-

- Multiple collisions at one time seem unrealistic at times. This can usually only be witnesses at the start of a game.
- Initial position of striker can overlap with other coins.

Treaps

Introduction :-

We have implemented sets and maps of C++ in racket using functional programming paradigm giving the best possible time complexity for the functions of these libraries. The data-structure which we use for it is treap. Treaps are randomised binary search trees i.e. unlike a simple BST, each node of a treap has two values, one bst value and one random value on which the treap is heap-ordered (wlog assume max-heap). So, our treap at every point of time maintains both the

invariants i.e. it is both bst-ordered (with respect to the BST value) and heap-ordered (wrt the random value). It is **highly probabilistic** for the height of the treap to be O (log n) regardless of the order of input unlike a simple BST where the height of the tree depends on the order of input. Hence, using treaps we can ensure that each of the set,map operations like insert,delete,search etc. can be done efficiently in O(log n) time. So what we aim to provide is two libraries which would do a handful of operations when they are imported using (provide "set.rkt") etc. These operations for "set.rkt" would include insert a number, delete a number, search a number, convert list into set and converse too, find union and intersections of two sets. For "map.rkt" we will provide operations like find the value associated with a key, insert a key-value pair, delete a key, convert map to list and converse.

Overall Idea of Design :-

Two main functions are:-

- **SPLIT**- It is a recursive function which given a value X splits a treap into two treaps so that all the values in the left treap are <=X and all the values in the right treap are >=X.
- **MERGE**-It merges two treaps into a single treap maintaining both the bst-invariant and the heap-invariant.

Other notable ideas used :-

- Pure functional programming paradigm.
- Concept of randomized data-structures and their efficiency.
- All the functions are implemented to the best possible time complexity i.e. O(log n) in most cases.

Abstractions :-

- While providing the libraries, we have just given the interface to the users hence abstracting all the helper functions and also the concept of a random value being clubbed with their data in such a way that it increases the efficiency of the functions.

- **Heavy code reusage-** Central to our code are two functions, namely split and merge which are re-used again and again in almost all the functions which we implemented including insert, delete.