Code Structure

The code for this game is structured in such a way as to separate functionality and threads into small concise functions. It is also built in such a way that the game mechanics can be altered easily by changing the defined constants and re-compiling the program. This makes it easy to adjust the game to be more playable but also allows us to easily test the game as we can alter the game in ways that make testing certain functionality much easier.

The following locks were used to ensure consistency of the data across all the threads:

- Mx : Lock for drawing to the screen using curses. Since we have one draw thread this is almost not needed however there are some points where allowing another thread to update parts of the screen on demand is useful.
- Saucers: Locks access to the saucers data structure. This ensures that saucers get updated properly as not all saucers updates come from the thread the saucer is running on. Some reads are also locked to ensure that certain reads are consistent with the most up to date information as we want things like collision detection to be as accurate as possible.
- Rockets: Performs the same functionality as saucers but for the rocket data structures.
- Reward: Performs the same functionality as saucers & rockets but for the rewards data structures.
- Escaped: Locks updating of the counter storing number of escaped saucers which is used to detect a possible end of game scenario. Reads are not locked as this number can only be incremented and updating of this number happens at a slower rate than other operations.
- CurrentRockets: Locks access to the counter storing how many rockets the user currently has. While this number can be incremented or decremented I do not lock the reads as if this number gets displayed improperly for a single screen refresh it is likely not to be noticed at all.
- currentPoints: Similar functionality to escaped and currentrockets except for the users points total.

The basic responsibilities for the program (excluding thread specific responsibilities) are as follows:

- Rockets, Rewards and Saucers handle updating their own position after sleeping for a specific amount of time however they do not draw themselves.

- Each time a rocket wakes up it checks for collision with a saucer and updates its position. If we detect a collision we can initiate the destroying of that saucer and the updating of the rocket count of the player
- Rewards are spawned by rockets upon detecting a collision with a 1/REWARDCHANCE of spawning (they do not spawn on every kill unless you set REWARDCHANCE to 1)
- Reward type is unknown to the user until after they pick it up. This is determined at collision time by the reward with a 1/ SPEEDUPCHANCE chance of spawning a rocket speed up while the rest are +5 rocket rewards.
- Saucers, rockets and rewards all have a struct associated with them for storing useful information about them like their current row/col. They also store a delay which determins how fast they move plus an alive flag. This allows us to know if the struct contains information about an alive entity. Saucers also have a die flag which if set tells the saucer it has been killed and it needs to clean up and exit.
- A game referee exists in order to monitor the game for harder to detect end game conditions. Simple ones that can be detected easily are done by other code however instances like 0 rockets + none on screen are end game conditions detected by this code.

Threads

The following threads and their responsibilities are used:

- Main: This is the game start point. Handles running the start code then handles user input for the rest of the program. This thread handles both 1p and 2p input as well as calling the cleanup code upon the user quitting the game.
- Saucer: Each saucer runs on its own thread. This thread sleeps for a pre determined amount of time (randomly chosen from a range) and upon waking up first checks if it is dead. If it is then it changes its alive state then exits. I decided not to externally kill the thread as a rocket kill could come in the middle of the saucers doing work. In order to avoid killing a thread in the middle of doing work I opted to flag a saucer for death and let it exit at an appropriate time. This ensures no threading issues as killing a saucer should not ever cause problems in the execution of the game. This thread locks all accesses to its saucer struct plus updating the escaped variable. The saucer also dynamically reduces its draw string as it hits the end of the screen in order to simulate it escaping out the side of the screen.

- Rocket: Each rocket runs on its own thread and is in charge of updating its location after each sleep/wake cycle. It also checks if it has collided by calling the rocket collision detection function. The collision function if it detects a collision handles killing the saucer as it knows which saucer it collided with. The rocket thread upon notification of a collision does a random roll for a reward spawn and then exits. All accesses to a rockets struct are locked.

- Reward: Much the same as rockets only they move in the opposite direction. They also look for collision with a turret and if they detect on they random roll for which kind of reward the user gets and they exit.

- spawnSaucer: A single thread handles spawning new saucers. It sleeps for a random specific interval of time and then spawns a new saucer thread with a random speed and location. This allows us to consolidate the handling of saucer spawning to its own thread so that it can focus on only doing this. The saucers data structure is locked while it is working as it scans for a free struct each time it wakes up.

- drawThread: As its name implies this thread focuses on drawing the screen. It sleeps for a very short interval to avoid spamming the screen with draws that are unnecessarily fasts. It loops through all the data structures for rockets, saucers and rewards and draws the alive ones. It does not lock its reads as the refreshes happen fast enough that a slightly dirty read should not even be noticeable. It does however lock the curses for the duration of its run meaning that a large portion of the time this thread has the curses lock. It does delegate turret drawing and the bottom screen info drawing to other functions because some other parts of the program need to draw these on demand. For example we re-draw the bottom screen info each time we have an escaped saucer (this could happen between screen refreshes).

- refereeThread: This thread handles checking the game for harder to detect end game conditions like out of rockets. Since being out of rockets is not enough to end the game it needs to make sure there are no rockets on screen as well. This thread does not sleep and constantly monitors the game.

Critical sections:

I attempted greatly to reduce the size of critical sections in this game to avoid conflict. I did specify an order for acquiring locks to avoid deadlocks however I do not believe there is more than 2 locks held simultaneously by a section, most are singles. Some of the reads are locked but most are not and there was an attempt to maintain only one section that updated a specific location leaving the rest as merely reads. The following critical sections exist:

- Rockets, Saucers and rewards each have to update their location upon wakeup. This section is very small and thus very quick.
- Updating of the global counters, rocket count, escaped saucers, etc, requires a lock and a short critical section.
- Long critical sections involve finding free structures for new rockets, saucers or rewards which requires locking one of the structure arrays and iterating through them all. These are likely the longest critical sections in the whole game as the rest are only a few lines at best.

Additional Features:

- Two player support
- Destroying saucers will sometimes yield rewards. Catch these rewards with your turret and get one of 2 rewards. 5 more rockets OR a rocket speedup.

UI and Controles

- To make the gam type ‘make’
- To start the game from the command line type ‘./saucer’.
- To start a 2 player game type ‘./saucer 2’
- Once in the game ‘S’ will start the game.
- The game should wait a short time before spawning saucers to ensure you are ready.
- **Player 1 controls:**

1. Left: ‘a’
2. Right: ‘d’
3. Fire: ‘w’ OR Space

- **Player 2 controls:**

1. Left: Left Arrow
2. Right: Right Arrow
3. Fire: Up Arrow

- Killing a saucer gives you 3 rockets
- A rocket reward is 5 rockets
- A speedup reward speeds up your rocket shots by a little more than double
- Game over happens when either too many saucers have left the screen (20 by default) OR if you have run out of rockets and have no currently in flight rockets or in flight rewards.
- Both players share the same rocket count and score.
- ‘Q’ will end the game at any time and bring you back to the command line

Design Decisions:

- Since 2 player mode is teamwork oriented both members share the same rocket count and score.
- The reward type is not known until you catch it as it makes coding different types of rewards easier since the reward need not have knowledge of what it is only decide what to do IF it gets caught. It also adds a nice element of surprise as you never know if missing a saucer for a reward catch will yield you that speed power up you have been looking for or just some rockets.
- Both players get the speed power up as it makes the code easier to implement.
- Lots of game definitions at the top of the code. This makes adjusting the game to your enjoyment or for testing more accessible. Can adjust rocket and reward speeds as well as how many escaped saucers we allow, how many lines at the top can saucers spawn on and how often rewards spawn plus how many of those rewards are rocket speedups.

Known issues or shortcomings:

- The code can currently handle 2 types of rewards but adding 3 or more would require a different system for determining which reward gets spawned.
- Both users cannot hold down a key and have the game repeatedly take in that input. Each user must repeatedly press their key which makes getting from one side of the screen to another more difficult.
- Get some blinking of the bottom info bar from time to time.