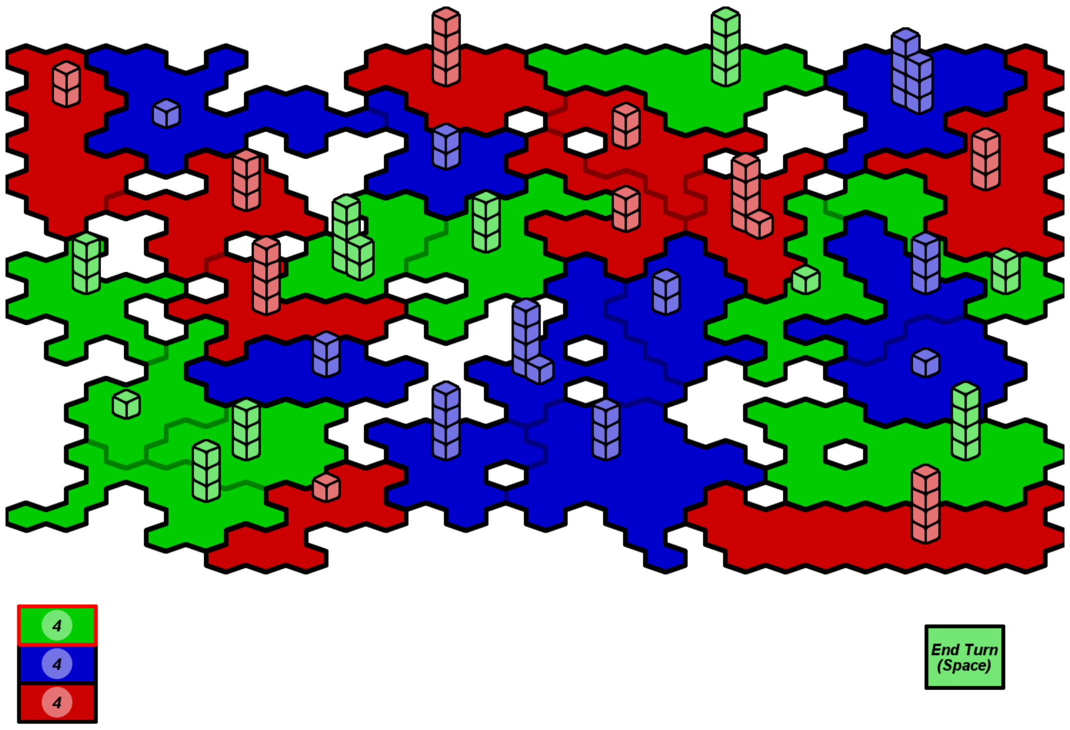
Block Battle

Block Battle is a turn-based strategy game similar to Risk, based on the flash game Dice Wars. The game is coded using Java and utilises Triple Buffering for frame rendering and an OOP structure which provides easy to follow logic for each game tick.



The game features a map generation algorithm yielding a new map on every play, and a simple AI for the user to play against. No Java game libraries or external assets are used, and all renders are calculated geometrically onto each frame.

1 How to Play

The game will start by distributing blocks, which act as units, to territories around the map. The player will always take their turn first. The section in the bottom left will tell the user the color of the current player, as well as all players current drafts, explained later.

N.B. You can hit the 'r' key to restart and generate a new map.

1.1 The Goal

The goal of the game is to control every territory on the board. This is done through attacking other players territories and invading their lands.

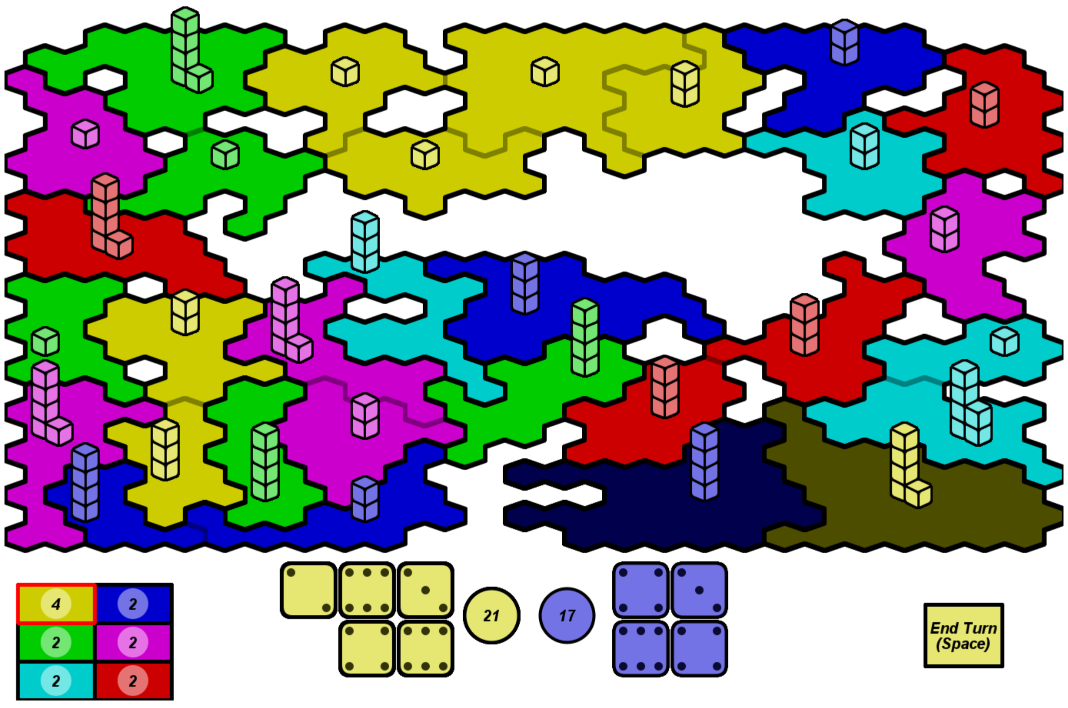
1.2 Taking Your Turn

First, click one of of territories which has more than one die, and then a neighbouring territory to battle it. Territories are considered neighbours when they have any hexagons touching.

The attacking and defending nations will each roll one die for each block in their territory. The player with the higher total sum of dice will win the battle. If the result is a draw, the defending player will win the battle.

If the attacker wins the battle, the defender will lose all blocks in the territory and the attacker will gain control of the territory, moving all blocks except one into it.

If the defender wins the battle, the attacker will lose all blocks in their attacking territory except one.



1.3 Drafting

When the player has finished their turn, they can hit the end turn button, or the space bar, to end their turn.

After a players turn ends they will automatically draft blocks into their territory. Blocks will be randomly distributed throughout their territories, up to a maximum of 8 blocks per territory. The number of blocks drafted is equal to the largest number of connected territories in their nation.

2 How it works

2.1 Program Structure

The program runs from a game loop in **Game**. The game loop makes sure that frames are rendered, whilst assuring that the appropriate number of game tick calculations happen between frames. A buffering strategy is used so frames run smoothly without flickering.

Game.class passes render and tick methods through classes to render objects and process game logic where is appropriate. A **Board** object is generated which processes board generation and directs flow of both game ticks and frame renders to territory and nation objects.

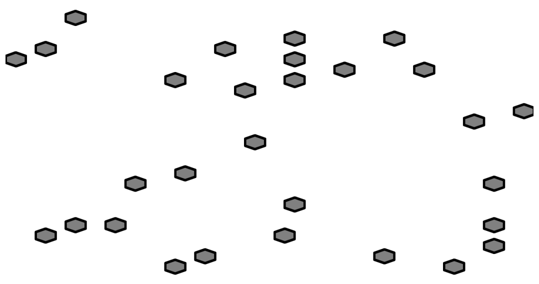
Territory objects each hold many **Hexagon** objects which correspond to the tiles within them, as well as a 'capital' hexagon where the blocks sit. **Territory** objects are responsible for logic exclusive to them, like the number of blocks in the territory, or which territories are neighbours. **Hexagon** objects render individually from a call from their territory.

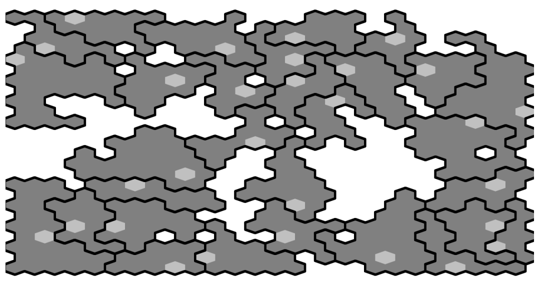
The **Mapper** class is responsible for much of the geometric calculations in frame rendering. This means processing hexagonal coordinates and returning pixel coordinates of hexagons and 3D blocks.

The **TurnHandler** processing the logic for player and computer turns. Ticks into this object are directed into different processes corresponding to the current stage of the turn.

2.2 Map Generation

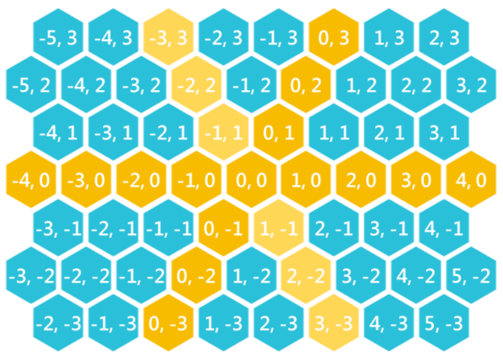
The map generation algorithm is very simple; hexagons are placed randomly across the coordinate grid, with weighting to be some distance away from other hexagons. The hexagons are then allowed to iteratively expand (where possible) into the space between them until all territories are connected, and have at least 2 neighbours (where possible).


(a) Initial Placement


(b) 80 iterations

2.3 Geometry

Much of the render calculations come down to simple trigonometry to calculate where the edges of hexagons and 3D cubes lie would lie on the pixel grid. More interestingly, are how you store hexagonal coordinates in a 2D system. Translations between hexagons can seem somewhat unintuitive in any system used.



The Axial coordinate system as above provides consistent translations between hexagons so was a good choice for this project.

3 How to Run

To compile and run navigate to **BlockBattle/src** and run **javac Game.java** then **java Game**