Simulating Kingdomino

Michael Wong and Sung Kang

1 Introduction

Kingdomino is a board game in which players attempt to create a five by five kingdom of tiles around the kingdom piece, a one by one tile, by using dominos (2×1 tiles) and connecting constrained by a set of rules. On each side of a domino, there is a unique color (or terrain type in the game) which can only be adjacent to domino side of the same color. There are six types of terrain types in this game: wheat fields, plains, water, swamps, forests, and mountains. In addition, each domino contains a number of crowns ranging from 0 to 3 and consists of either 2 of the same or different terrain types. The score is determined by counting the number of adjacent tiles with the same terrain type and multiplying this by the number of crowns within that area. The player with the highest score wins.

From a mathematical perspective, the game is full of questions regarding the various combinations of tile placements and the points associated with each placement. We will address some of the questions through the implementation of Python programming language.

In this paper, we aim to answer the following questions:

- How many unique terminal board states are there?
- What is the maximum number of points possible?
- How many ways are there to tessellate a 5×5 board in Kingdomino and in general?

2 Dominos and Tessellations

In geometry, a tessellation is a tiling of a 2D surface with polygons such that no part of the surface is left exposed. Our polygons in this case are 2×1 rectangles, and the surface to cover is an $m\times n$ grid of unit squares. In this paper we will refer to this type of grid as a *board*.

It is well known that $m\times n$ boards can be tessellated in

$$\prod_{j=1}^{\frac{n}{2}}\prod_{k=1}^{\frac{m}{2}}(4cos^2\frac{\pi j}{n+1}+4cos^2\frac{\pi k}{m+1})$$

unique ways, for m and n even [3]. If we constrain m and n to be the same (but still even) we have that a $2n\times2n$ board can be tessellated in

$$2^n(2k+1)^2$$

unique ways [3]. Thus for $n=1,2,3,4,\ldots$ there are $2,36,6728,12988816,\ldots$ unique tilings respectively. Note that in this case, two tessellations that are rotationally symmetric are not considered equivalent tilings. In other words, relative orientation matters. If we were to consider rotationally symmetric tessellations equivalent, we can reduce the number of tessellations by a factor of 4 (note that the figures listed above for lower values of n are all divisible by 4).

n must be even in order to ensure that the board can be tessellated. This is trivial to show; an $2n+1 \times 2n+1$ board would contain $(2n+1)^2$ unit squares. The square of any odd number is itself odd, and so is not a multiple of two. Thus it cannot be tiled by dominos which each take up two cells of the grid, i.e. there will always be one or more squares of the board left exposed. In Kingdomino, the game avoids incomplete tilings by adding a starting square — the castle — a single unit square. Thus, it is possible to tessellate a $2n+1 \times 2n+1$ board with $\lfloor \frac{2n+1}{2} \rfloor$ dominos and 1×1 square. However, as far as we know, a relationship between n and the number of tessellations possible is an open problem. Later in this paper, we propose and implement a brute force algorithm to find all tessellations for a $2n+1 \times 2n+1$ board.

3 Problem Scope and Framing

We can apply this information to Kingdomino in an attempt to approximate the size of the game's state space, that is the space of all possible (and reachable) states of the game board, i.e. the game tree. Before we can do that, let's first view the problem in a simpler frame.

Kingdomino is usually played with four, three or two players. However, for the purposes of this analysis, we view the problem as a one player game: drawing random dominos from 48 total, add dominos to a 5×5 board. The game ends when the player finds a tessellation or cannot put down another tile. Additionally, we do not count bonus scoring schemes like awarding fully tessellated boards, etc.

Finding a maximum scoring tessellation can easily be done with a non-deterministic algorithm in polynomial time (and later we will propose such an algorithm) and thus is in the class NP. We suspect the problem is NP-complete, though a proof of this via reduction is beyond the scope of this paper. In either case, the size of the game tree is exponential in n.

The size of the game tree is a bit difficult to calculate. Gedda et. al puts it at roughly 10^{61} [2] (note that this is for four players). We can try to approximate by analyzing the state space as a tree: there is a maximum depth of 18. The maximum breadth of the tree is trickier. Imagine a filled 3×3 board embedded in the top-left corner of a 5×5 board. To add an additional domino, we can add it 6 ways on the bottom rows vertically, 6 ways on the rightmost columns horizontally, then and additional 6 ways horizontally on the second to last row and 6 ways horizontally on the second to last column. In total, that is an upper bound of 24. Thus the tree has a maximum number of leaves roughly 24^{18}.

Because of this large breadth, a breadth-first search technique would quickly exceed the memory of any modern computer. To limit the amount of memory necessary, we use a depth-first search technique to traverse the game tree.

4 Computational Approach

We represent each board as a 5×5 matrix. Each tile (1×1 square) has a type, encoded as an integer 0-6, and a number of crowns (0 by default). Squares are grouped in pairs corresponding to the 48 game dominos.

We give our algorithm for enumerating the states of the game tree and finding the maximum scoring board. Note the algorithm for placing the starting 1×1 tile is not shown.

As we discussed in the previous section, the game tree for a 5×5 board is incredibly large. On a 1GHz processor, it would take a few months to run the computation. Even with modern processors and massive parallelism, one might hope to complete the computation in a few weeks. However, the algorithm has room for some optimizations.

Algorithm 1 Kingdomino$_{DFS}(b, r, M)$

```
states ← 0
tessellations ← 0
maxb ← b
if board b is terminal then
    if board b is a tessellation then
        return (1,1, maxb)
    else
        return (1, 0, maxb)
    end if
end if
for domino d ∈ remaining domino set r do
    P ← {next states when d is placed on b}
    for board b' ∈ P do
        if b' ∉ visited states M then
            (c,t,mb) = Kingdomino_DFS(b', r − {d}, M)
            M ← M + b'
            states ← states + c
            tessellations ← tessellations + t
            if score of mb > score of maxb then
                maxb ← mb
            end if
        end if
    end for
end for
return (states, tessellations, maxb)
```

Note that placing the starting 1×1 tile derives 25 possible states. If we consider rotationally symmetric tilings equivalent, we can reduce this by a factor of 4. Indeed, the starting tile only needs to be placed in $\lceil \frac{2n+1}{2} \rceil \lfloor \frac{2n+1}{2} \rfloor + 1$ cells as shown in Figure 1.

Additionally, we can take advantage of overlapping subproblems to reduce compute time. For example, if I have two dominos A and B and place them in two different places in the order B then A or A then B, they result in the same state but represent distinct branches of computation with near identical subtrees. By tracking the states already explored, we can prune subtrees from our DFS.

Even with these optimizations, the game tree is still too large to feasibly explore within a short amount of time. So, we reduce Kingdomino to a 3×3 board. This gave us tangible results within a relatively short amount of time (1 hour).

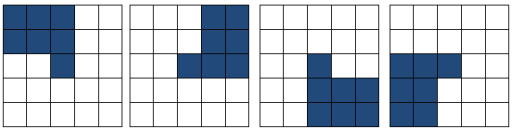

Figure 1: The starting 1×1 tile for a 5×5 board yields unique states in only 7 starting cells, every other cell is rotationally symmettric to one of these 7

When we apply the rules of Kingdomino for a 3×3 board, then there were 989421 possible end states with 801973 complete tilings. The maximum possible score for a 3×3 board is 36 as shown in Figure 2.

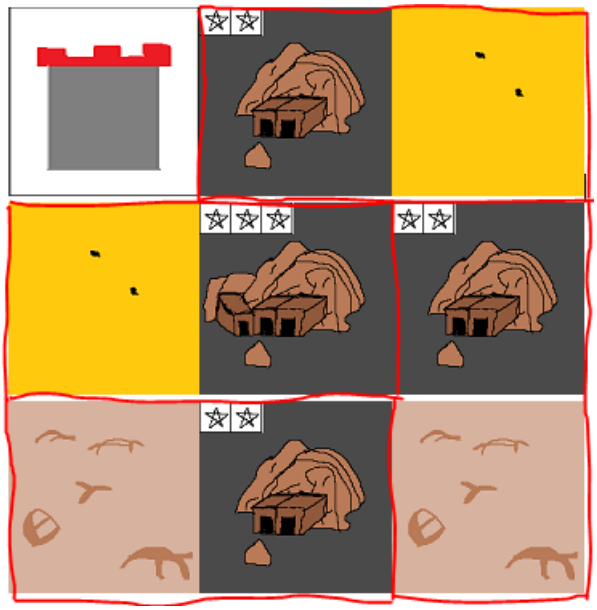

Figure 2: Maximum score configuration for a 3×3 Kingdomino board, which yields a score of 27.

5 General $2n+1 \times 2n+1$ Boards

Similar to Kingdomino, we use a general $n \times n$ matrix to represent a board. Instead of 48 distinct dominos, we only have two kinds: horizontal and

vertical. All horizontal dominos are encoded with a 1, vertical dominos with a 2, starting tiles with a 3 and empty spaces with a 0. Note that we can encode a board as a number — a concatenation of n^2 two digit binary numbers for each cell. Thus a board of size of $n \times n$ can have 2^{2n^2} different numbered representations. We use the following algorithm to enumerate all tilings of a general $n \times n$ board.

Algorithm 2 Domino$_{DFS}(b, M)$

$states \leftarrow 0$
$tessellations \leftarrow 0$
if board b is terminal **then**
  if board b is a tessellation **then**
    return (1,1)
  else
    return (1, 0)
  end if
end if
$P \leftarrow \{$next states when d is placed on $b\}$
for board $b' \in P$ **do**
  if $b' \notin$ visited set M **then**
    (c,t) = Domino$_{DFS}(b', M)$
    $states \leftarrow states + c$
    $tessellations \leftarrow tessellations + t$
    $M \leftarrow M + \{b'\}$
  end if
end for
return $(states, tessellations)$

Note that we only use the 1×1 starting tile if the board is of odd dimensions. For a sanity check, we run our algorithm on a 4×4 board and get a correct tessellation count of 36, pictured in Figure 3.

For a 3×3 board, there were 144 terminal tilings, out of those 24 were tessellations. For a 5×5 board, there were 303,388 terminal tilings with 2,768 tessellations. For the same reasons we could not completely explore a 5×5 Kingdomino board, tiling a 7×7 board was too computationally expensive to simulate. While there are infinitely man curves through those two points, the increase seems to be exponential from 24 to 2,768.

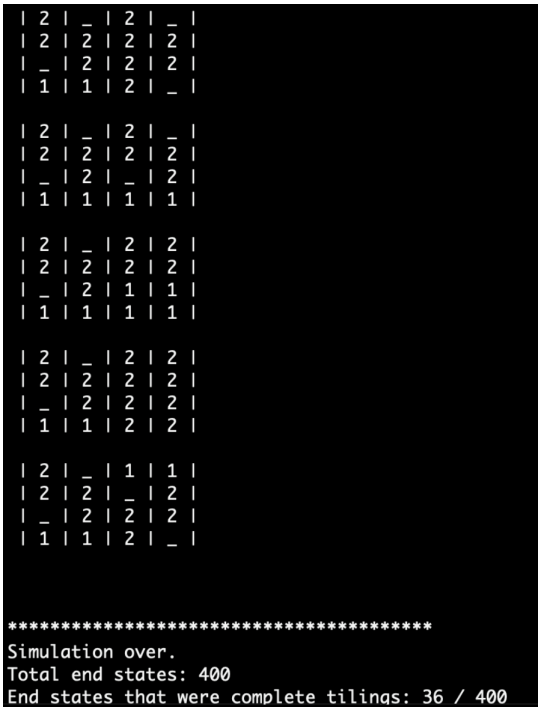

Figure 3: Screenshot of simulation from the terminal window showing possible tilings for a 4×4 board. Note that the result is what we predicted with the equation given in section 2.

6 Conclusion

Through the analysis of Kingdomino and general domino tilings, we have developed an algorithm to simulate domino tilings for general $n \times n$ boards with and without Kingdomino rules. With better hardware, it is theoretically possible to simulate Kingdomino tilings for boards with dimensions $2n+1 \times 2n+1$ for $n \geq 1$. The program is open-source and available on GitHub here: `www.github.com/michaelwong2/Kingdomino`.

7 References

1. Cathala, Bruno. *Kingdomino*. Blue Orange Games, 2016.

2. Gedda M.; Lagerkvist M. Z.; M. Butler. Monte Carlo Methods for the Game Kingdomino. 2018 IEEE Conference on Computational Intelligence and Games (CIG), Maastricht, 2018, 1-8.

3. Pachter, L. Combinatorial Approaches and Conjectures for 2-Divisibility Problems Concerning Domino Tilings of Polyominoes. *The Electric Journal of Combinatorics*. **1997**, *4*.