# **2048**

# 20 (Slightly) Different 2048 VersionsProject Summary

The idea of the game of 2048 is that you move and merge tiles with equivalent values to generate tiles with larger values. The grid on which you can move the tile is a 4 \* 4 grid. Every time you can move in one of the four directions: up, down, left, right. Once a tile with value 2048 is being created, you win; if there is a deadlock, which means there are no possible ways to further make a move or merge, you lose the game. We would like to model the logic behind this game. Particularly we would like to model the mechanics of movements. We will be diving into the status of each tile before and after movements, whether they can move, and toward which direction they are moving.

# Propositions

TODO

* Bring in the figure for the edge labels
* Make sure they are numbers and not letters
* Finish rewriting to mirror the names used in the code

Draft:

# location (tile, loc): tile t is at location loc, the location is where on the board that things exist

# adjacent(tile, orientation): check if there are adjacent tiles near t\*

# empty (tile, orientation): check if adjacent places are empty

# is\_empty (loc): check if a location loc is empty

# can\_merge (tile, orientation): check if the tile can merge

# row\_can\_move (row): check if at least 1 move can be made in that row

# column\_can\_move (column): check if at least 1 move can be made in that column

# able\_2\_move (tile, orientation): check if at least 1 move can be made on the given orientation

# random (loc): after a move, a random new tile is generated at an location

# get (nextTile, orientation): check the next tile in the given orientation, if the next tile is of the same value, return true

# row\_or\_column\_can\_move(orientation): check if a row or column can move

# Constraints

TODO:

* Go over the proposal constraints and refine for what’s currently done
* Add the new constraints that came into the project since then
  + Just summarize the super simple ones (e.g., no self loops)

Draft:

* No two tiles can be put to the same location. For tiles t1 and t2 with location loc:
* location(t1, loc) \/ location(t2, loc)
* If a move can be made and there the next tile in the given orientation is as of the same value. then they can merge.
* get (tile, orientation) /\ able\_2\_move (orientation) can\_merge (tile, orientation)
* no new tile can be generated on a not empty location
* ¬ is\_empty (loc)  ¬ random (loc)
* If the next tile in the given orientation is as of the same value, if the next tile in the direction is empty, then we can move.
* get (nextTile, orientation) \/ empty (tile, orientation)  able\_to\_move (tile, orientation)
* If at least 1 out of 4 tiles on the same row or column (depending on the orientation of movement, i.e. we use row if move left/right, use column if move up/down) can move, we are able to make at least 1 move in that orientation
* able\_to\_move (t1, orientation) \/ able\_to\_move (t2, orientation) \/ able\_to\_move (t3, orientation) \/ able\_to\_move (t4, orientation)  row\_or\_column\_can\_move (orientation)
* Only 1 tile is generated randomly amoung the all the empty locations.
* IDK how to write it

# Model Exploration

## Sth to be elaborated more on

In our previous proposition, we used propositions U and L to declare the direction we are moving. However, we noticed that these propositions may be too hard to be used in the constraints and make things more complicated. Therefore, in our new propositions, we removed U and L. Instead, we used the parameter orientation to declare the direction.

In our previous proposition, to determine if two tiles can merge, we must check if, in the given direction, the adjacent next tile has the same value as the original one or the tile that has the same value on the row/column (depending on the direction) it is on, and the spaces between them are empty. We believe that this is a bit complicated. To simplify this, we came up with a new proposition, the get (nextTile, orientation) function, it checks the next tile in the given direction to see if it matches the original tile.

Differentiating each tile from another

Previously, we were using a dictionary to model our grid with the coordinates as keys and their numerical values as values. This turned out to be problematic as many of the constraints and propositions require us to check the coordinates. It would be hard to check the coordinates, so instead of them directly being the keys, we are creating tiles as objects, which would then be inserted as keys into the dictionary.

# Jape Proof Ideas

We haven’t started this yet.

# Requested Feedback

1. *How are the propositions and constraints of our model? We have been told that we need to focus more on the mechanics of tile movements on the feedback we received earlier, and we have tried hard to modify our propositions and constraints to make them more valid.*

# First-Order Extension

*Describe how you might extend your model to a predicate logic setting, including how both the propositions and constraints would be updated.* ***There is no need to implement this extension!***

Nope, just haven’t started this yet.