D2 - D5 FINAL SUBMISSION

Status	Done
Date	@December 7, 2023 11:59 PM
Tags	Deliverable

Project Summary

In Tetris, players manipulate falling shapes called Tetromino's within a playing area measuring ten squares in width and twenty squares in height. There are seven distinct Tetromino shapes. The objective is to manoeuvre and rotate these falling pieces to complete entire rows in the playing field. This process is referred to as a line clear.

The aim of this project is to model a game of Tetris, and analysing the state of the game so that given a random board and Tetromino in play we can find if a row can be cleared successfully within 20 moves of the Tetromino.

For purposes of this project we denote cells as belonging to the board, and blocks as belonging to Tetromino's, i.e., a cell on the board is occupied, or a Tetromino is composed of blocks. However, when a Tetromino cannot move down any further, its blocks turn into cells.

Propositions

Board Propositions

- $c_{x,y}$ - true if the cell at coordinate x (0-9), y (0-19) is occupied, note we will be using matrix notation for these moves, i.e starting (0, 0) in the top left.
- Rc - true if a row has been cleared, i.e victory

Tetromino Propositions

- $b_{x,y,t}$ - true if the block at coordinate (x, y) at time t exists. A Tetromino is composed of blocks.
- $t_{x,y,s,r,t}$ - true if Tetromino defined by x, y, s, r, and t exists. Each subscript means as follows:
  - x - the x coordinate of the anchor*
  - y - the y coordinate of the anchor*
  - s - the Tetromino's shape
  - r - the Tetromino's rotation
  - t - what tick time is currently on

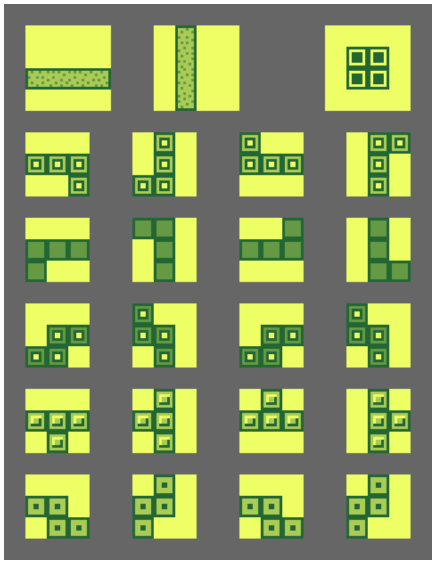
* We employ the use of an anchor to encapsulate a multi-coordinate Tetromino into one coordinate. The anchor coordinate is the point of rotation for a Tetromino.

In Tetris there are 7 tetromino's. When creating our propositions we pass in a keyword for the Tetromino we want to create, namely one of:

- line
- square
- J
- L
- S
- T

- Z

The respective images of the Tetromino's listed above are depicted in this diagram.



Tetromino's and their rotations

Each tetromino also has 4 rotations, though not all 4 rotations are unique.

Since Tetromino's are multi-coordinate objects we need to encapsulate them into one object to make things easier. We've set up lists of extensional values that define each Tetromino for each rotation. Take the line Tetromino as an example in it's original position, its extensional values look like this:

```
[(0, 0), (1, 0), (2, 0), (-1, 0)]
```

So, when the Tetromino is formed at coordinate (x,y), the whole Tetromino's blocks are created as such:

```
Block 1: (x + 0, y + 0)
Block 2: (x + 1, y + 0)
Block 3: (x + 2, y + 0)
Block 4: (x + -1, y + 0)
```

Constraints

Board Constraints

- If all cells in a row all filled, then the row can be cleared
  - Rc is true if every p_x for any y is true, eg: $(p_{0,y} \land p_{1,y} \land \cdots \land p_{9,y}) \rightarrow Rc$
- A Tetromino can only have 1 legal x coordinate per row
  - $t_{0,y,\text{line},0,0} \lor t_{1,y,\text{line},0,0} \lor ... \lor t_{9,y,\text{line},0,0}$
- A block cannot overlap with an occupied cell
  - $\neg(b_{0,0,0} \land c_{0,0})$

Tetromino Constraints

The following examples are for the T Tetromino unless otherwise specified. The T Tetromino has its anchor at the cross-section

- A Tetromino will drop if there are no cells below it. A successful drop will increase time by one up to 20 ticks.
  - $(t_{1,0,\mathrm{T},0,0} \wedge \neg(c_{0,2} \vee c_{1,2} \vee c_{2,2})) \rightarrow t_{1,1,\mathrm{T},0,1}$
- A Tetromino will turn into cells if there is a cell below it
  - $(t_{1,0,\mathrm{T},0,0} \wedge (c_{0,2} \vee c_{1,2} \vee c_{2,2})) \rightarrow (c_{1,0} \wedge c_{0,1} \wedge c_{1,1} \wedge c_{2,1})$

Time Constraints

- If a Tetromino goes beyond the allowed 20 ticks, then the board is not clearable
  - $t_{0,0,\mathrm{T},0,20} \rightarrow \neg Rc$

Model Exploration

Algorithm

Below are the phases/algorithm our project takes.

1. Row candidates
   - This initial function scans every row as a candidate for potential clearing. It looks for single contiguous holes (as we cannot clear a line with more than 1 hole) of 1-4 cells wide (as we cannot clear a line with a hole greater than 4 cells wide). The function adds constraints that this set of Tetromino's (whatever they may be) definitely cannot clear the row. It serves as an early return case where if we do get a black listed Tetromino we can know immediately a row cannot be cleared.
2. Free column
   - Since we do not allow a Tetromino to move horizontally while falling, an unobstructed column down towards the hole in the row must exist in order for the Tetromino to fall into it and clear the row. It serves as an early return case where if we do have a cell blocking the Tetromino's path, then we can know immediately a row cannot be cleared.
3. Support
   - Like in the real world, things fall in Tetris. Given a row we want to clear, we also have to make sure that there exists cells underneath the hole to allow the Tetromino to sit. It serves as an early return case where if we do have at least one cell in accordance with the shape of the Tetromino underneath the row's hole, then we can know immediately a row cannot be cleared.
4. The fall
   - Lastly, after all these checks have been made, we know a hole that the Tetromino fits in exists, there is a free path down to it, and there is something to support the Tetromino. The final step is for Bauhaus to brute-force and try to create encoding which will model the Tetromino's path down to the hole and clear the row

We figured that implementing pruning phases would help guide the program along a bit and make solving such problems easier.

Board

We initially wanted a board of size 10x40 but found that is not the standard game of Tetris' size according to the Tetris guidelines[1]. We moved to a 10x20 version initially but the size isn't often consistent in terms of dropping location on the y-axis, so discussion is still required on the exact size and semantics of the board.

Reducing the size of the board obviously reduces the number of propositions needed, not only for the board itself, but for many of the Tetromino constraints. For example, there are less cells to check for overlaps, or less rows to check if they're cleared.

Tetromino's

We initially wanted to implement this project more akin to a true game a Tetris. Namely elements like shifting and rotating the Tetromino while falling which allow it to bypass overhangs or rotate through chucks of cells. During that time we tried to implement an A* algorithm to find a path down from the spawn point to the hole in the row. We inevitably decided to bring back our focus to a more simpler game by taking out functionalities like those mentioned above.

Jape Proof Ideas

We will use propositions such as the following:

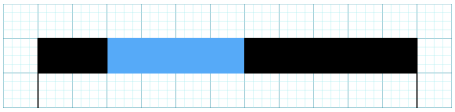
- Pc00 to mean that coordinate (0,0) is occupied by a cell
- Pt00 to mean that coordinate (0,0) is occupied by a block
- Prc to mean that a row is cleared

Tetromino Suitability

> 💡 Given a row with a hole 4 cells wide and a line Tetromino, the row can be cleared if the line is horizontal, but the row cannot be cleared if the line is vertical.

In Tetris, choosing the right rotation for a Tetromino is a crucial aspect of the game. These two proofs intend to demonstrate this concept.

Clearable



(Pc00∧Pc10∧Pc20∧Pc30∧Pc40∧Pc50∧Pc60∧Pc70∧Pc80∧Pc90)→Prc

- If all cells in the row are occupied then the row is cleared

Pt20∧Pt30∧Pt40∧Pt50

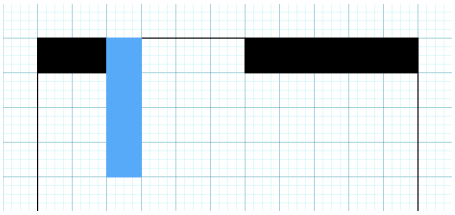
- The horizontal line Tetromino

Ptxy→Pcxy

- Tetromino blocks will turn into cells

1:	(Pc00∧Pc10∧Pc20∧Pc30∧Pc40∧Pc50∧Pc60∧Pc70∧Pc80∧Pc90)→Prc	premise
2:	Pt20∧Pt30∧Pt40∧Pt50, Pt20→Pc20, Pt30→Pc30, Pt40→Pc40	premises
3:	Pt50→Pc50, Pc00∧Pc10∧Pc60∧Pc70∧Pc80∧Pc90	premises
4:	Pc90	∧ elim 3.2
5:	Pc00∧Pc10∧Pc60∧Pc70∧Pc80	∧ elim 3.2
6:	Pc00∧Pc10∧Pc60∧Pc70	∧ elim 5
7:	Pc70	∧ elim 6
8:	Pc00∧Pc10∧Pc60	∧ elim 6
9:	Pc60	∧ elim 8
10:	Pc00∧Pc10	∧ elim 8
11:	Pc10	∧ elim 10
12:	Pc00	∧ elim 10
13:	Pc00∧Pc10	∧ intro 12,11
14:	Pc80	∧ elim 5
15:	Pt50	∧ elim 2.1
16:	Pc50	→ elim 3.1,15
17:	Pt20∧Pt30∧Pt40	∧ elim 2.1
18:	Pt40	∧ elim 17
19:	Pc40	→ elim 2.4,18
20:	Pt20∧Pt30	∧ elim 17
21:	Pt30	∧ elim 20
22:	Pc30	→ elim 2.3,21
23:	Pt20	∧ elim 20
24:	Pc20	→ elim 2.2,23
25:	Pc00∧Pc10∧Pc20	∧ intro 10,24
26:	Pc00∧Pc10∧Pc20∧Pc30	∧ intro 25,22
27:	Pc00∧Pc10∧Pc20∧Pc30∧Pc40	∧ intro 26,19
28:	Pc00∧Pc10∧Pc20∧Pc30∧Pc40∧Pc50	∧ intro 27,16
29:	Pc00∧Pc10∧Pc20∧Pc30∧Pc40∧Pc50∧Pc60	∧ intro 28,9
30:	Pc00∧Pc10∧Pc20∧Pc30∧Pc40∧Pc50∧Pc60∧Pc70	∧ intro 29,7
31:	Pc00∧Pc10∧Pc20∧Pc30∧Pc40∧Pc50∧Pc60∧Pc70∧Pc80	∧ intro 30,14
32:	Pc00∧Pc10∧Pc20∧Pc30∧Pc40∧Pc50∧Pc60∧Pc70∧Pc80∧Pc90	∧ intro 31,4
33:	Prc	→ elim 1,32

Unclearable



- We assume that the board is the same as in the *Clearable* proof

Pt20∧Pt21∧Pt22∧Pt23

- The vertical line Tetromino

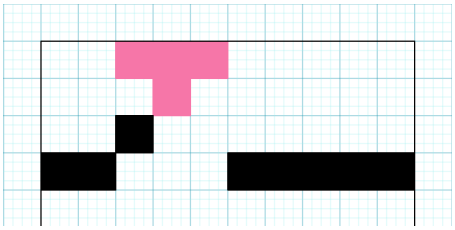
1:	Pt20∧Pt21∧Pt22∧Pt23	premise
2:	(Pt20∧Pt21∧Pt22∧Pt23)→¬Prc	premise
3:	¬Prc	→ elim 2,1

This proof is different due to the fact our project contains pruning functions. One such function finds all contiguous holes in every row and determines if the given Tetromino can fit in that hole. In this case we have a hole which is 4 blocks wide and a vertical line which is only one block wide. So, the function adds the constraint that a vertical line cannot clear the row

Overhang

> 💡 Given a row with a hole 3 cells wide and a T Tetromino, the row cannot be cleared since their is a cell blocking it's path.

In this project horizontal movement while falling is not allowed. If there are any cells present within the column above the hole the Tetromino cannot move around it and will thus hit it. Therefore, it is not possible to clear the row.



Pt20∧Pt30∧Pt40∧Pt31

- The T Tetromino

(Pc20∨Pc30∨Pc40∨Pc21∨Pc31∨Pc41∨Pc22∨Pc32∨Pc42)→¬Prc

- If any cells are filled within the column, then the row cannot be cleared

Pc22∧Pc03∧Pc13∧Pc53∧Pc63∧Pc73∧Pc83∧Pc93

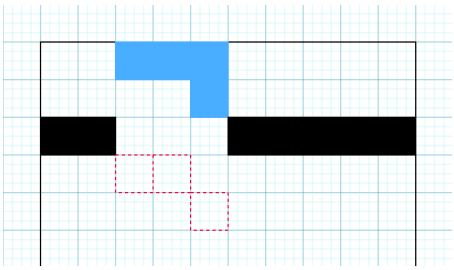
- The cells that are filled

1:	Pt20∧Pt30∧Pt40∧Pt31, (Pc20∨Pc30∨Pc40∨Pc21∨Pc31∨Pc41∨Pc22∨Pc32∨Pc42)→¬Prc, Pc22∧Pc03∧Pc13∧Pc53∧Pc63∧Pc73∧Pc83∧Pc93	premises
2:	Pc22∧Pc03∧Pc13∧Pc53∧Pc63∧Pc73∧Pc83	∧ elim 1.3
3:	Pc22∧Pc03∧Pc13∧Pc53∧Pc63∧Pc73	∧ elim 2
4:	Pc22∧Pc03∧Pc13∧Pc53∧Pc63	∧ elim 3
5:	Pc22∧Pc03∧Pc13∧Pc53	∧ elim 4
6:	Pc22∧Pc03∧Pc13	∧ elim 5
7:	Pc22∧Pc03	∧ elim 6
8:	Pc22	∧ elim 7
9:	Pc20∨Pc30∨Pc40∨Pc21∨Pc31∨Pc41∨Pc22	∨ intro 8
10:	Pc20∨Pc30∨Pc40∨Pc21∨Pc31∨Pc41∨Pc22∨Pc32	∨ intro 9
11:	Pc20∨Pc30∨Pc40∨Pc21∨Pc31∨Pc41∨Pc22∨Pc32∨Pc42	∨ intro 10
12:	¬Prc	→ elim 1.2,11

Support Needed

> 💡 Given a row with a hole 3 cells wide and an L Tetromino, the row cannot be cleared since their does not exist cells underneath the row.

In Tetris a Tetromino will fall until it meets a cell below it. This proof intends to demonstrate that even though we have a hole 3 cells wide and a 3 block-wide Tetromino, we cannot clear that row because there are no cells underneath to support the Tetromino.



Pt20∧Pt30∧Pt40∧Pt41

- The L Tetromino

(¬Pc23∨¬Pc33∨¬Pc44)→¬Prc

- If there is not at least one cell to support the L Tetromino, then the row cannot be cleared

Pc02∧Pc12∧¬Pc22∧¬Pc32∧¬Pc42∧Pc52∧Pc62∧Pc72∧Pc82∧Pc92∧¬Pc03∧¬Pc13∧¬Pc23∧¬Pc33∧Pc43∧¬Pc531¬Pc

- The cells that are filled and not filled

1:	Pt20∧Pt30∧Pt40∧Pt41, (¬Pc23∨¬Pc33∨¬Pc44)→¬Prc	premises
2:	Pc02∧Pc12∧¬Pc22∧¬Pc32∧¬Pc42∧Pc52∧Pc62∧Pc72∧Pc82∧Pc92∧¬Pc03∧¬Pc13∧¬Pc23∧¬Pc33∧¬Pc43∧¬Pc53 ∧¬Pc63∧¬Pc73∧¬Pc83∧¬Pc93∧¬Pc04∧¬Pc14∧¬Pc24∧¬Pc34∧¬Pc44∧¬Pc54∧¬Pc64∧¬Pc74∧¬Pc84∧¬Pc94	premise
3:	Pc02∧Pc12∧¬Pc22∧¬Pc32∧¬Pc42∧Pc52∧Pc62∧Pc72∧Pc82∧Pc92∧¬Pc03∧¬Pc13∧¬Pc23∧¬Pc33∧¬Pc43 ∧¬Pc53∧¬Pc63∧¬Pc73∧¬Pc83∧¬Pc93∧¬Pc04∧¬Pc14∧¬Pc24∧¬Pc34∧¬Pc44∧¬Pc54∧¬Pc64∧¬Pc74∧¬Pc84	∧ elim 2
4:	Pc02∧Pc12∧¬Pc22∧¬Pc32∧¬Pc42∧Pc52∧Pc62∧Pc72∧Pc82∧Pc92∧¬Pc03∧¬Pc13∧¬Pc23∧¬Pc33∧¬Pc43 ∧¬Pc53∧¬Pc63∧¬Pc73∧¬Pc83∧¬Pc93∧¬Pc04∧¬Pc14∧¬Pc24∧¬Pc34∧¬Pc44∧¬Pc54∧¬Pc64∧¬Pc74	∧ elim 3
5:	Pc02∧Pc12∧¬Pc22∧¬Pc32∧¬Pc42∧Pc52∧Pc62∧Pc72∧Pc82∧Pc92∧¬Pc03∧¬Pc13∧¬Pc23∧¬Pc33 ∧¬Pc43∧¬Pc53∧¬Pc63∧¬Pc73∧¬Pc83∧¬Pc93∧¬Pc04∧¬Pc14∧¬Pc24∧¬Pc34∧¬Pc44∧¬Pc54∧¬Pc64	∧ elim 4
6:	Pc02∧Pc12∧¬Pc22∧¬Pc32∧¬Pc42∧Pc52∧Pc62∧Pc72∧Pc82∧Pc92∧¬Pc03∧¬Pc13∧¬Pc23∧¬Pc33 ∧¬Pc43∧¬Pc53∧¬Pc63∧¬Pc73∧¬Pc83∧¬Pc93∧¬Pc04∧¬Pc14∧¬Pc24∧¬Pc34∧¬Pc44∧¬Pc54	∧ elim 5
7:	Pc02∧Pc12∧¬Pc22∧¬Pc32∧¬Pc42∧Pc52∧Pc62∧Pc72∧Pc82∧Pc92∧¬Pc03∧¬Pc13∧¬Pc23 ∧¬Pc33∧¬Pc43∧¬Pc53∧¬Pc63∧¬Pc73∧¬Pc83∧¬Pc93∧¬Pc04∧¬Pc14∧¬Pc24∧¬Pc34∧¬Pc44	∧ elim 6
8:	¬Pc44	∧ elim 7
9:	¬Pc23∨¬Pc33∨¬Pc44	∨ intro 8
10:	¬Prc	→ elim 1.2,9

First-Order Extension

Our project lends itself very well to predicate logic, due to the fact that the most commonly used propositions are the x and y cells, which have much in common with each other. Below is a list of propositions and constraints translated to predicate logic:

Domain of Discource

- Natural numbers (used for coordinates and time)
- Objects for Tetromino shapes

Predicates

- `Block(x, y, t)`: Block of a Tetromino with coordinate (x, y) is occupied on the board at time t
- `Tetromino(a, s, r, t)`: Tetromino of shape s and rotation r is located at coordinate a (tuple of x and y) at time t
- `Cell(x, y)`: Cell with coordinate (x, y) is occupied on the board
- `Row_Cleared()`: There exists a row which can be cleared on the board

Constraints

- Whether a row is cleared

- $\exists y.(\forall x.(\text{Block}(x, y)))$

- A Tetromino is only allowed to occupy 1 legal x coordinate in a row
  - $\forall y.\forall s.\forall r.\forall t.\exists x.(\text{Tetromino}((x, y), s, r, t))$
- A block of a Tetromino cannot overlap with a cell
  - $\forall x.\forall y.\forall t.(\neg(\text{Block}(x, y, t) \land \text{Cell}(x, y)))$
- A Tetromino will drop by 1 y coordinate if there are no cells below it
  - $\forall x.\forall y.\forall s.\forall r.\forall t.((\text{Tetromino}((x, y), s, r, t) \land \neg\text{Cell}(x, y + 1)) \rightarrow \text{Tetromino}((x, y + 1), s, r, t + 1))$
  - It should be noted that we have not added every cell that needs to be check as it varies depending on the Tetromino (in the code it does). For simplicity we've include one cell to make the point. In reality there could be 2-4 cells to check for vacancy.
- A Tetromino will turn into cells if there exists at least one cell below
  - $\forall x.\forall y.\forall s.\forall r.\forall t.((\text{Tetromino}((x, y), s, r, t) \land (\text{Cell}(x, y + 1) \lor ...)) \rightarrow (\text{Cell}(x, y) \land ...)$
  - It should be noted that we have not added every cell that needs to be checked or created as it varies depending on the Tetromino (in the code it does). For simplicity we've include one cell to make the point. In reality there could be 2-4 cells to check for vacancy, and 4 cells to create.

References

1.

https://tetris.wiki/Tetris_Guideline