GRP_11: Modelling Street Fighter

Marcus Chiang (19MHZC)

Jalal Ghaus (20SJG)

Sophie Ellwood (19SE)

Pratyush Kaintura (20PK1)

Course Modelling Project

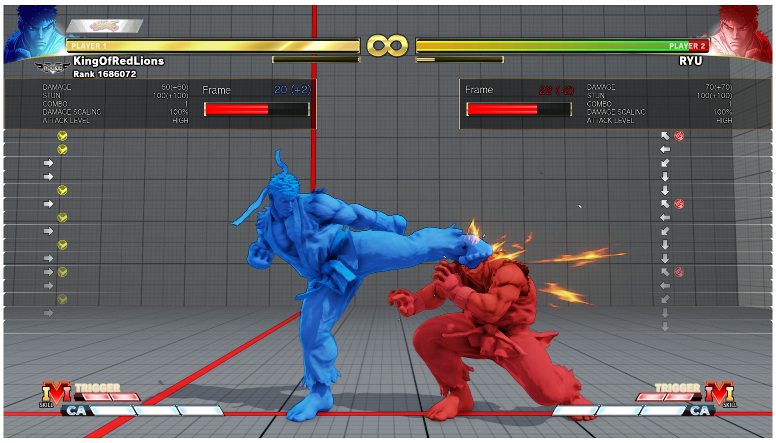
CISC/CMPE 204

Logic for Computing Science

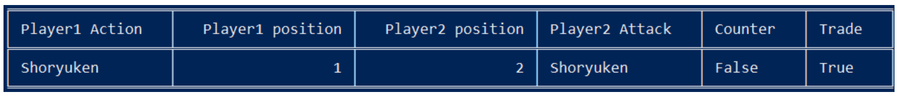
December 14, 2021

Abstract

Our project is designed to loosely model the game Street fighter. The program will analyze the current circumstances of the situation and find solutions based on distance, attack-speed, and the general rules of Street Fighter. We examine one ”snapshot” of the fight at a time in order to set up the model with the correct propositions and to be able to evaluate the configuration using SAT solvers.



The output of the model will detail the positions of the two players as well as the attacks and best defensive move(s) in a table.



Player1 Action	Player1 position	Player2 position	Player2 Attack	Counter	Trade
Shoryuken	1	2	Shoryuken	False	True

We have included features such range of attacks and frames per second (for measuring speed of moves) which will be explained in the proposition and model exploration sections. We have omitted features such as a health bar and back-and-forth playing.

Propositions

Many of our propositions span between each player where the specified player is either 1 or 2 and is followed by an underscore. For example, Player 1 doing a crouch kick would be $crouchKick_1$. We have listed the propositions without the player specified as the propositions apply to both.

Each action proposition has 3 parameters that come into play during a fight. For example, the overhead punch has the parameters overhead punch, 2, 22, and overhead. The first parameter denotes the name of the attack. The second parameter indicates the range of that attack, so an overhead punch has a 2 tile range. The third parameter indicates the speed (frames) of the attack, so a lower number indicates a faster attack. (lower frames) Overhead punch would take 22 frames so it would be slower than a light punch as that attack takes 3 frames. Lastly an attack has a specific area it attacks. It can be overhead, mid, or low. An overhead would attack at the player's head, mid would attack the stomach and chest area, a low attack would hit their legs. Speed and area of attack will be explained below the propositions list.

- Trade denotes that attacks from both players have landed
- A throw break is counted as a trade, if a player is thrown, they can throw back to trade with the other player.
- lightP denotes a player performing a light punch (2 tiles, 3 frames, mid)
- overheadP denotes a player performing a overhead punch (2 tiles, 22 frames, overhead)
- crouchK denotes a player performing a crouch kick (3 tiles, 6 frames, low)
- standK denotes a player performing a standing kick (4,8, mid)
- T denotes a player performing a throw (1 tile, 5 frames, unblockable)
- H denotes a player performing a hadouken (7 tiles, 14 frames, mid)
- SHORYU denotes a player performing a shoryuken (1 tile, 0 frames, mid). The move has a 0 frame startup since shoryuken is an invincible attack from the beginning of its start-up.
- HB denotes a player performing a high block
- LB denotes a player performing a low block
- NJUMP denotes a player performing a neutral jump
- FJUMP denotes a player performing a forward jump
- MB denotes a player is performing a mid bloc
- C denotes a player has countered an attack

Blocking high blocks any overhead/ mid attacks such as light punch, overhead punch, or a standing kick. Blocking low blocks low attacks such as crouch kick. When a player is blocking high, a crouch kick could land and if the player is blocking low then a light punch, standing kick, etc. would land. However a throw attack CANNOT be blocked so no matter what block is being utilized, a throw will land provided players are within range. If both players use throw then a **throw break** occurs where neither players are affected.
Lastly if both players use the same attack on each other when the attack is in range, it causes both of the attacks to land which is given the term trade.

A player performing a counter depends on the attack's speed. For example, let's say player 1 and player 2 are adjacent to each other, and player 1 does a light punch while player 2 does a crouch kick at the same time. A light punch takes 3 frames compared to a crouch kick which takes 6 frames, therefore player 1's attack is faster. This means that the light punch will land instead of the crouch kick, and the C_1 proposition will be true as the punch countered the kick.

We have classes that create position object for each player. Each player's respective position objects are stored in p1PositionArray or p2PositionArray. Each object from the classes take one argument: an integer that is used to calculate distance in later functions.
We have 2 arrays that contain every possible move that can be performed in the situation. The lists are called p1ActionArray and p2ActionArray, which distinguish player 1's moves and the player 2's moves.

To help visualize the model: here are some common attacks from the actual Street Fighter game:

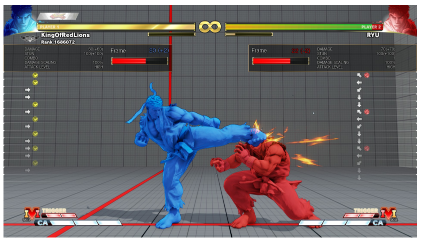


Figure 1: Standing Kick

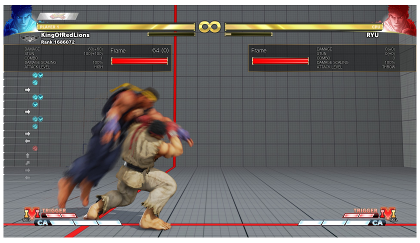


Figure 2: Throw

Figure 3: Hadouken

Figure 4: Punch

Constraints

Attack Range:

- Players must adhere to whatever range their move has, if punch has a range of 1, they must have at most one space between them. $(\neg(P_1 \wedge \neg adjacent))$
- If the distance between two players is greater than the range of one of their moves, restrict that situation $(\neg(p1position \wedge p2position \wedge action))$
- Nothing happens if player blocks an attack $(((P_1 \vee K_1 \vee H_1 \vee SHORYU) \wedge (B \vee_2)) \implies bothNeutral)$

Attack Constraints:

- If the attack of player 1 is faster than player2, it will be a counter hit $(\neg(counterVar \wedge p1attack \wedge p2attack))$

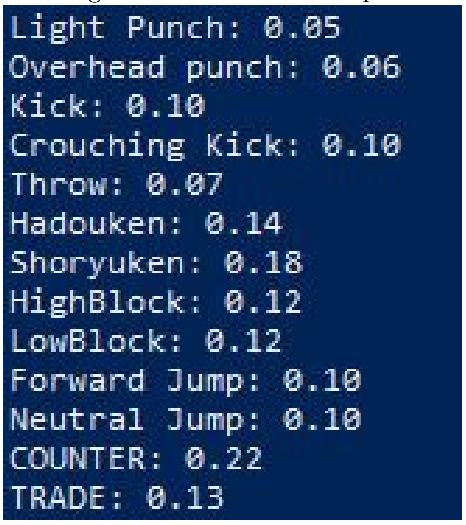
- If both players use the same move on each other, it will trade $\neg(tradeVar \wedge p1attack \wedge p2attack)$
- Player 1 must block, jump, or dp a punch $(\neg(P_2 \wedge \neg(B_1 \vee JUMP_1 \vee SHORYU_1)))$
- Player 2 may not punch and kick at the same time $(\neg(P_2 \wedge K_1))$
- Player 1 may block, jump, or dp a kick $(\neg(K_2 \wedge \neg(B_1 \vee JUMP_1 \vee SHORYU_1)))$
- -Player 1 must throw back if thrown $(\neg(T_2 \wedge \neg(T_1 \vee P_1)))$
- Player 1 may not dp a throw $(\neg(T_2 \wedge SHORYU_1))$
- Player 1 must jump, block, fire their own hadouken, or stuff theirs $(\neg(H_2 \wedge \neg(JUMP_1 \vee B_1 \vee H_1 \vee SHORyU_1 \vee K_1 \vee P_1)))$
- Player 1 must throw or block a shoryuken $(\neg(SHORYU_2 \wedge \neg(T_1 \vee B_1)))$

Model Exploration

Debugging: Debugging the code relied heavily on the likelihood function. Printing a solution only gave us one solution of thousands, and it would consistently produce variations of the same solution. Understanding the likelihood and seeing how it resembled actual Street Fighter told us whether or not we were approaching an accurate model.

Screenshot of different explorations:

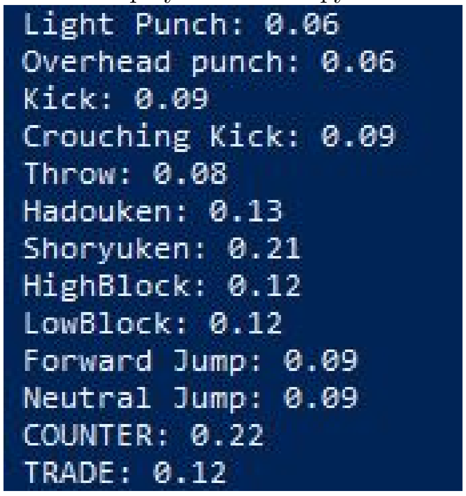
Figure 5: The normal output



```
Light Punch: 0.05
Overhead punch: 0.06
Kick: 0.10
Crouching Kick: 0.10
Throw: 0.07
Hadouken: 0.14
Shoryuken: 0.18
HighBlock: 0.12
LowBlock: 0.12
Forward Jump: 0.10
Neutral Jump: 0.10
COUNTER: 0.22
TRADE: 0.13
```

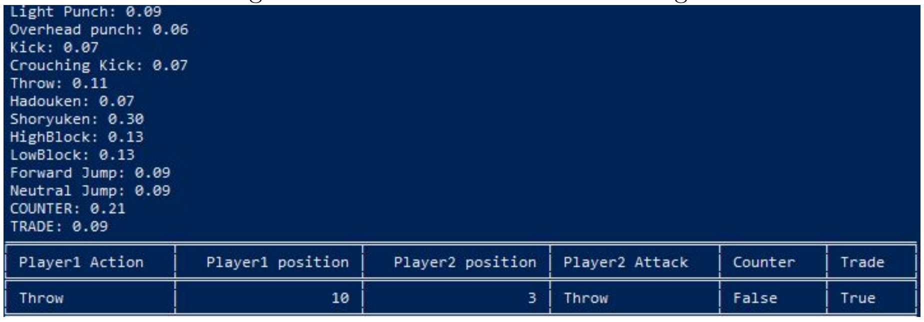
The normal output with no drastic changes greatly resembles traditional Street Fighter. Long ranged pokes like kicks are incredibly valuable, with hadoken also being a common move to attack from afar. Shoryuken is also a valuable move, as it is used in defence to beat close range moves and jumps. High blocking and low blocking are equally as valuable, as there is a 50/50 chance of being hit with an overhead attack or low attack. Counter hits are also fairly common, but not overwhelmingly so; in a normal game, they would lead to greater combo potential. Throws have a smaller proportion, but this is because their use is limited to defending against shoryuken and trading with an opponents throw. Light punch and overhead punch are the lowest frequency, as they are used as either a fast poke in the close range, or a slow overhead that can be counter-hit easily.

Figure 6: Both players can occupy the same spot


```
Light Punch: 0.06
Overhead punch: 0.06
Kick: 0.09
Crouching Kick: 0.09
Throw: 0.08
Hadouken: 0.13
Shoryuken: 0.21
HighBlock: 0.12
LowBlock: 0.12
Forward Jump: 0.09
Neutral Jump: 0.09
COUNTER: 0.22
TRADE: 0.12
```

Both players occupying the same spot leads to more situations in which close range pokes are valuable. The proportion of hadoken and kicks are lowered, with more favour being given to fast close range moves like light punch, throws, and shoryuken.

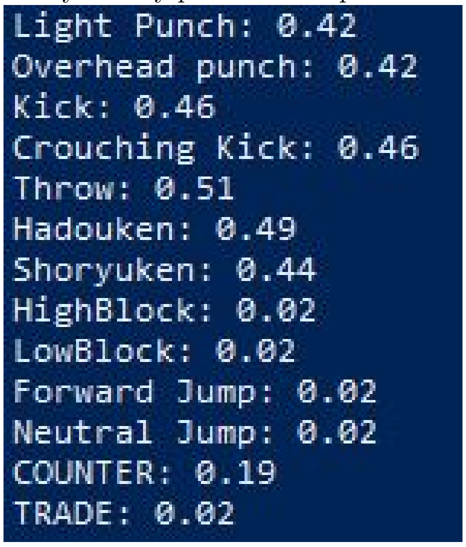
Figure 7: No constraints on attack range


```
Light Punch: 0.09
Overhead punch: 0.06
Kick: 0.07
Crouching Kick: 0.07
Throw: 0.11
Hadouken: 0.07
Shoryuken: 0.30
HighBlock: 0.13
LowBlock: 0.13
Forward Jump: 0.09
Neutral Jump: 0.09
COUNTER: 0.21
TRADE: 0.09
```

Player1 Action	Player1 position	Player2 position	Player2 Attack	Counter	Trade
Throw	10	3	Throw	False	True

Player 1 performs a throw from 7 spaces away (absurd), one of the reasons we decided to not randomize attacks and have the player be smarter about range.

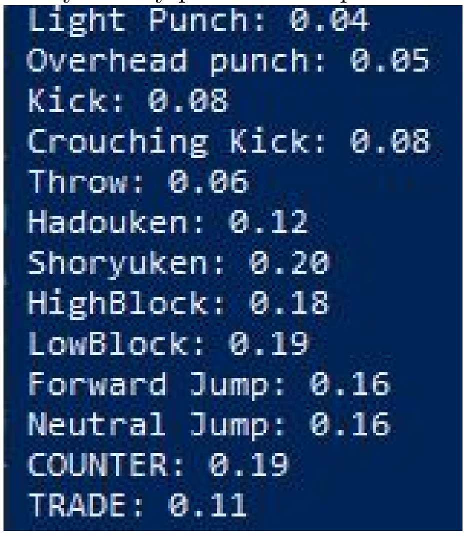
Figure 8: Player1 may perform multiple attacks at once



```
Light Punch: 0.42
Overhead punch: 0.42
Kick: 0.46
Crouching Kick: 0.46
Throw: 0.51
Hadouken: 0.49
Shoryuken: 0.44
HighBlock: 0.02
LowBlock: 0.02
Forward Jump: 0.02
Neutral Jump: 0.02
COUNTER: 0.19
TRADE: 0.02
```

The proportions for attacks are insanely higher and dilute the other actions. The trade variable also has a lower proportion since introducing all these combinations of attacks lowers the number of combinations of each player performing the same attack.

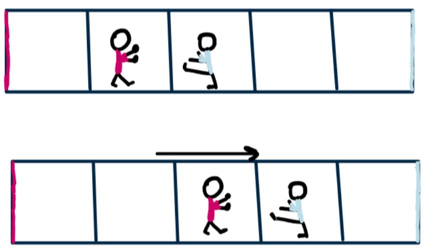
Figure 9: Player1 may perform multiple actions at once



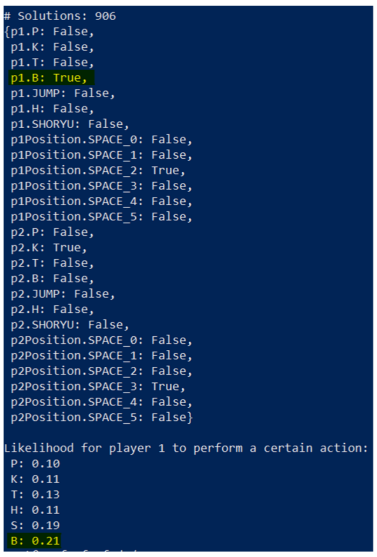
```
Light Punch: 0.04
Overhead punch: 0.05
Kick: 0.08
Crouching Kick: 0.08
Throw: 0.06
Hadouken: 0.12
Shoryuken: 0.20
HighBlock: 0.18
LowBlock: 0.19
Forward Jump: 0.16
Neutral Jump: 0.16
COUNTER: 0.19
TRADE: 0.11
```

Similarly to multiple attacks at once, the greater proportion of possible actions dilutes the proportion of attacks. This leads the program to believe that jumping is much more frequent, making the shoryuken anti-air more valuable. Counter hit and trades are less frequent since the player is more prone to being airborne and blocking.

Based on feedback we received, we were exploring the option of adjusting our project into a back-and-forth game where two players would compete on a 2D board with the winner of a match advancing a square and the loser retreating a square (see below).
However, this would require us to randomize blocking and hit detection. We tried this by randomizing the truth value of blocking, as well as using the position propositions and ranges to add constraints to all instances of a player not getting hit. This led to a completely random back and forth of two player just doing whatever it takes to hit each other. We found that the likelihoods did not match traditional street fighter. Shoryukens were incredibly dominant and the players used short ranged moves when on opposite sides of the stage. We decided to create a flawed defence, but one that still understood space and basic defence.



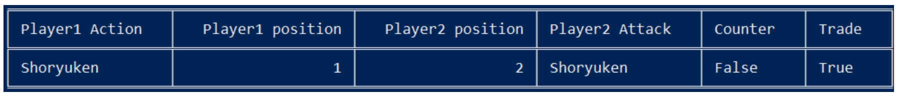
At one point, we had player two's actions being different every time when our program was being run, but player 1's defence was always a block. This is technically a correct solution as block is a versatile move and is a highly likely solution, however we would have liked to have gotten more variation in defenses. We imposed/removed(?) constraints to adjust the likelihoods of each move meaning that when the program was run multiple times, the actions shown had a higher chance of being different.



```
# Solutions: 906
{p1.P: False,
 p1.K: False,
 p1.T: False,
 p1.B: True,
 p1.JUMP: False,
 p1.H: False,
 p1.SHORYU: False,
 p1Position.SPACE_0: False,
 p1Position.SPACE_1: False,
 p1Position.SPACE_2: True,
 p1Position.SPACE_3: False,
 p1Position.SPACE_4: False,
 p1Position.SPACE_5: False,
 p2.P: False,
 p2.K: True,
 p2.T: False,
 p2.B: False,
 p2.JUMP: False,
 p2.H: False,
 p2.SHORYU: False,
 p2Position.SPACE_0: False,
 p2Position.SPACE_1: False,
 p2Position.SPACE_2: False,
 p2Position.SPACE_3: True,
 p2Position.SPACE_4: False,
 p2Position.SPACE_5: False}

Likelihood for player 1 to perform a certain action:
 P: 0.10
 K: 0.11
 T: 0.13
 H: 0.11
 S: 0.19
 B: 0.21
```

As seen above, our output consisted of a large dictionary of all possible actions and their corresponding truth values. This format made it difficult to read which values are actually true so we created a table consisting only of the values that were true (the important ones). After doing this, our table clearly shows the player attack, player defense, the two positions of the players as well as whether the defense resulted in a counter or trade.



Player1 Action	Player1 position	Player2 position	Player2 Attack	Counter	Trade
Shoryuken	1	2	Shoryuken	False	True

What we added to improve complexity:

- High and Overhead blocking
- Frame data (Attack start-up speed)
- Counter hits and trades
- Expanded the stage and gave attack ranges more variety
- Gave the program more sophisticated range calculating capabilities
- Added unblockable moves

First-Order Extension

Predicates:
p1Attack(a): object a is an attack by player 1
p1Action(b): object b is an action by player 1
p1Interaction(c): object c is an interaction by player 1
p2Attack(d): object d is an attack by player 2
p2Action(e): object e is an action by player 2
p1Position(f): object f is a position which player 1 can occupy
p2Position(g): object g is a position which player 2 can occupy

Constraints:
PLAYER LIMITATIONS
For all player one positions, for all player two positions, negate the conjunction of any x.
$\forall .y \neg (p1Position(y) \wedge p2Position(y))$

For all player one attacks, negate any two variables
$\forall x, y(p1Attack(x) \wedge p1Attack(y))$

For all player one actions, negate any two variables
$\forall x \neg (p1Attack(x) \wedge p1attack(x)$

For all player one attacks, there exists a block, negate the conjunction of all player one attacks and block
$\forall x \exists y \neg(p1Attack(x) \wedge p2attack(y))$

There exists a jump and there exists a block, negate the conjunction of the two as a player may not jump and block at the same time.
$\exists .jump \exists .block \neg(p1Action(jump) \wedge p1Action(block))$

p1Attack(x) = a player one attack, p1Action(y) = some jump
For all x.(negate P(x) and y (the jump variable) $\forall x(\neg(P(x) \wedge p1Action(jump)))$

do the same for negating any attack and block
$\forall attack. \exists block. \neg(p1Attack(attack) \wedge p1Action(block)$

RANGE
For all player attacks, for all player 1 positions, for all player 2 positions, there exists an attack which has range that is less than the absolute value of the difference between the two positions.
$\forall p1positions, p2positions. \exists attackrange \neg(p1Attack(attackrange < (p1positions - p2positions)) \wedge p1Positions(p1positions) \wedge p2Positions(p2positions)$

COUNTERHIT
For all player 1 attacks, for all player 2 attacks, there exists a p1attack startup which is greater than p2attack startup, negate this with p1 counterhit.
$\forall xz \neg(p1Attack(z) \wedge p2Attack(x) \wedge) p1Counterhit(y)$

TRADE
For all player 1 attacks, for all player 2 attacks, there exists an attack which is not of equal speed between the two players, negate that in conjunction with trade variable
$\forall p1Attacks \forall p2Attacks(\neg(\exists faster(p1Attacks, p2Attacks)) \vee trade)$

HIGH/LOW
For all p2 attacks, there exists an attack with the overhead property, negate this in conjunction with low blocking
$\forall player2attacks. \exists overhead. \exists lowBlock \neg(p2Attack(overhead) \wedge p1Action(lowBlock))$

$\forall player2attacks. \exists low. \exists highBlock \neg(p2Attack(low) \wedge p1Action(highBlock))$

$\forall player2attacks. \exists kick. \exists highBlock \neg(p2Attack(kick) \wedge p1Action(highBlock))$

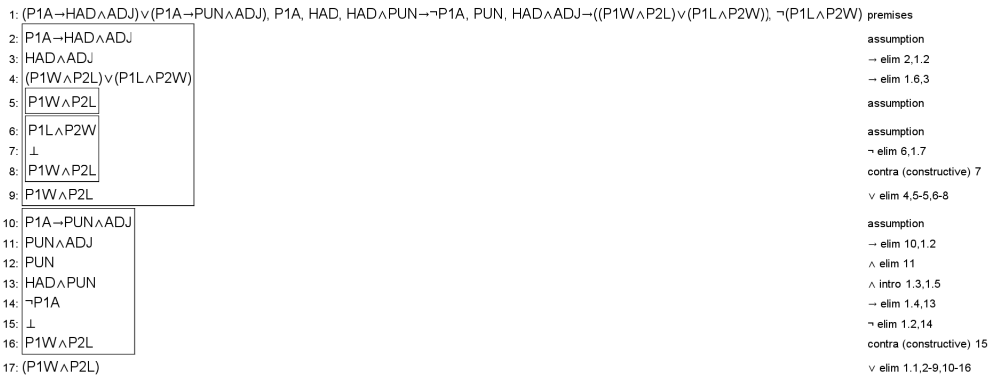
We have used predicate logic in our project by implementing embedded for loops. Embedded for loops could be represented by ∀, and one example of this is for range constraints. We have said that for all positions, player 1 and player 2 can not be in the same position. We can represent this logically: $\forall . x \neg (p1Position(x) \wedge p2Position(x))$.

Embedded for loops have been an integral part of our project as they have made creating constraints simpler. Another example of predicate logic is when talking about possible attacks. We say that for all possible attacks there exists a defense. We can represent this logically: $\forall x \exists y (p1Attack(x) \wedge p2defence(y))$.

Jape Proofs

Proof 1
Proposition Symbols:

- P1A - Player 1 has a valid attack
- HAD - A Hadouken attack
- PUN - A Punch attack
- ADJ - Players are adjacent
- P1W - Player 1 Wins
- P1L - Player 1 Loses
- P2W - Player 2 Wins
- P2L - Player 2 Loses



```
 1: (P1A→HAD∧ADJ)∨(P1A→PUN∧ADJ), P1A, HAD, HAD∧PUN→¬P1A, PUN, HAD∧ADJ→((P1W∧P2L)∨(P1L∧P2W)), ¬(P1L∧P2W) premises
 2: P1A→HAD∧ADJ                 assumption
 3: HAD∧ADJ                     → elim 2,1.2
 4: (P1W∧P2L)∨(P1L∧P2W)         → elim 1.6,3
 5:  P1W∧P2L                    assumption
 6:  P1L∧P2W                    assumption
 7:  ⊥                          ¬ elim 6,1.7
 8:  P1W∧P2L                    contra (constructive) 7
 9: P1W∧P2L                     ∨ elim 4,5-5,6-8
10: P1A→PUN∧ADJ                 assumption
11: PUN∧ADJ                     → elim 10,1.2
12: PUN                         ∧ elim 11
13: HAD∧PUN                     ∧ intro 1.3,1.5
14: ¬P1A                        → elim 1.4,13
15: ⊥                           ¬ elim 1.2,14
16: P1W∧P2L                     contra (constructive) 15
17: (P1W∧P2L)                   ∨ elim 1.1,2-9,10-16
```

Simple Explanation:
Given that player 1 has a valid attack with a Hadouken and player one does NOT lose, then player one wins.

Detailed Explanation:
If player one has a valid attack, this attack consists of a Hadouken and being adjacent to the other player OR a punch and being adjacent to the other player.

We also have the constraint that a punch and Hadouken could not be done at the same time therefore any attempt would not be a valid attack. If player one does a Hadouken while being adjacent, either player 1 will win and player 2 will lose OR player 1 will lose and player 2 will win.
We are given that player one does a valid attack, that this attack is a Hadouken and that player 1 does NOT lose.

Proof 2
Proposition Symbols:

- PK - A Kick attack
- BLO - A block defense
- PUN - A punch defense
- PMJ - A jump defence
- DEF - A valid defensive move
- ADJ - Players are adjacent
- 1SP - Players have one space between them
- 2SP - Players have two spaces between them

```
 1: PK, DEF, (PK→ADJ)∧(PK→1 SP)∧(PK→¬2 SP) , DEF→((PUN∧ADJ)∨(BLO∧1 SP)∨(PMJ∧2 SP))  premises
 2: (PK→ADJ)∧(PK→1 SP)                                   ∧ elim 1.3
 3: PK→ADJ                                               ∧ elim 2
 4: PK→1 SP                                              ∧ elim 2
 5: PK→¬2 SP                                             ∧ elim 1.3
 6: ¬2 SP                                                → elim 5,1.1
 7: (PUN∧ADJ)∨(BLO∧1 SP)∨(PMJ∧2 SP)                      → elim 1.4,1.2
 8: | (PUN∧ADJ)∨(BLO∧1 SP)                               assumption
 9: | | PUN∧ADJ                                          assumption
10: | | PUN                                              ∧ elim 9
11: | | PUN∨BLO                                          ∨ intro 10
12: | | BLO∧1 SP                                         assumption
13: | | BLO                                              ∧ elim 12
14: | | PUN∨BLO                                          ∨ intro 13
15: | PUN∨BLO                                            ∨ elim 8,9-11,12-14
16: | PMJ∧2 SP                                           assumption
17: | 2 SP                                               ∧ elim 16
18: | ⊥                                                  ¬ elim 17,6
19: | PUN∨BLO                                            contra (constructive) 18
20: PUN∨BLO                                              ∨ elim 7,8-15,16-19
```

Simple Explanation:
Given a kick attack that works when players are one space apart or closer, the defenses are solutions that work within one space or closer (punch and block).

Detailed Explanation:
We are given that both an attacking kick and defensive move occur. The kick is valid when players are adjacent or within one space and the defensive moves work when adjacent, 1 space or two spaces apart respectively. Trying to use a defense that works 2 spaces away with an attack that works within 1 space

causes a contradiction therefore the solutions are the punch and block.

Proof 3
Proposition Symbols:

- PM - A defensive middle block
- PL - A defensive low block
- PH - A defensive high block
- PJ - An offensive jump attack
- PK - An offensive kick attack
- PP - An offensive punch attack
- PK - An offensive kick attack

```
 1: PH→¬(PM∨PL) , PJ→¬PK∧¬PP , (PK∧PL)∨(PP∧PM)∨(PJ∧PH) , PJ   premises
 2: ¬PK∧¬PP                          → elim 1.2,1.4
 3: ¬PP                              ∧ elim 2
 4: ¬PK                              ∧ elim 2
 5: | (PK∧PL)∨(PP∧PM)                assumption
 6: | | PK∧PL                        assumption
 7: | | PK                           ∧ elim 6
 8: | | ⊥                            ¬ elim 7,4
 9: | | | PM∨PL                      assumption
10: | | | ⊥                          hyp 8
11: | | ¬(PM∨PL)                     ¬ intro 9-10
12: | | PP∧PM                        assumption
13: | | PP                           ∧ elim 12
14: | | ⊥                            ¬ elim 13,3
15: | | | PM∨PL                      assumption
16: | | | ⊥                          hyp 14
17: | | ¬(PM∨PL)                     ¬ intro 15-16
18: | ¬(PM∨PL)                       ∨ elim 5,6-11,12-17
19: | PJ∧PH                          assumption
20: | PH                             ∧ elim 19
21: | ¬(PM∨PL)                       → elim 1.1,20
22: | | PM∨PL                        assumption
23: | | ⊥                            ¬ elim 22,21
24: | ¬(PM∨PL)                       ¬ intro 22-23
25: ¬(PM∨PL)                         ∨ elim 1.3,5-18,19-24
```

Simple Explanation:
This proof deals with the idea of low, high and mid blocks. If a jump attack is performed, the best move would be to block with an attack that defends from high moves therefore a high block.

Detailed Explanation:

First off, if a high block is being done, there cannot be a low block or mid block happening at the same time. We also know that a kick is a ”low” attack, a punch is a ”mid” attack and a jump is a ”high” attack. Given that the attack is a jump we also decide that the attack cannot be a punch or kick. The assumption boxes go through each possibility of attack paired with the idea of being low middle or high to conclude that the best defense is NOT the mid or low defenses.
NOTE: It is note entirely necessary to include the two assumption boxes from lines 9-10 and 15-16 but they are there to help show how the contradiction in the larger assumption boxes affects the conclusion.

Summary

Street fighter is a binary game, you are performing an attack, and every attack has its unique set of implications. What you are doing dictates your other actions and what your opponent may do. We used this to centralize our computations and output on player 1, which having player 2 pick one of any actions for player 1 to deal with. Attacks have different properties (speed, range, how you block it, etc...), we modelled this by creating attack propositions and adjusting the fields of the object to hold the attack properties. We then created propositions for the possible positions on the stage that player 1 and player 2 may occupy. This began using raw propositional logic and formulas, but evolved to include the object oriented aspect of bauhaus. With these fundamentals in place, we could start implementing more advanced features of Street Fighter, like counter hits and trades. We could also implement the concept of high/low blocking and mix-ups, which is a crucial element of every fighting game.
Our program accurately represents and calculates the most difficult part of Street Fighter as a skilled, but imperfect player would. The bot still trades hits with the opponent, but understands when to block, how to block, when to jump, and when to take a risk and attack back.