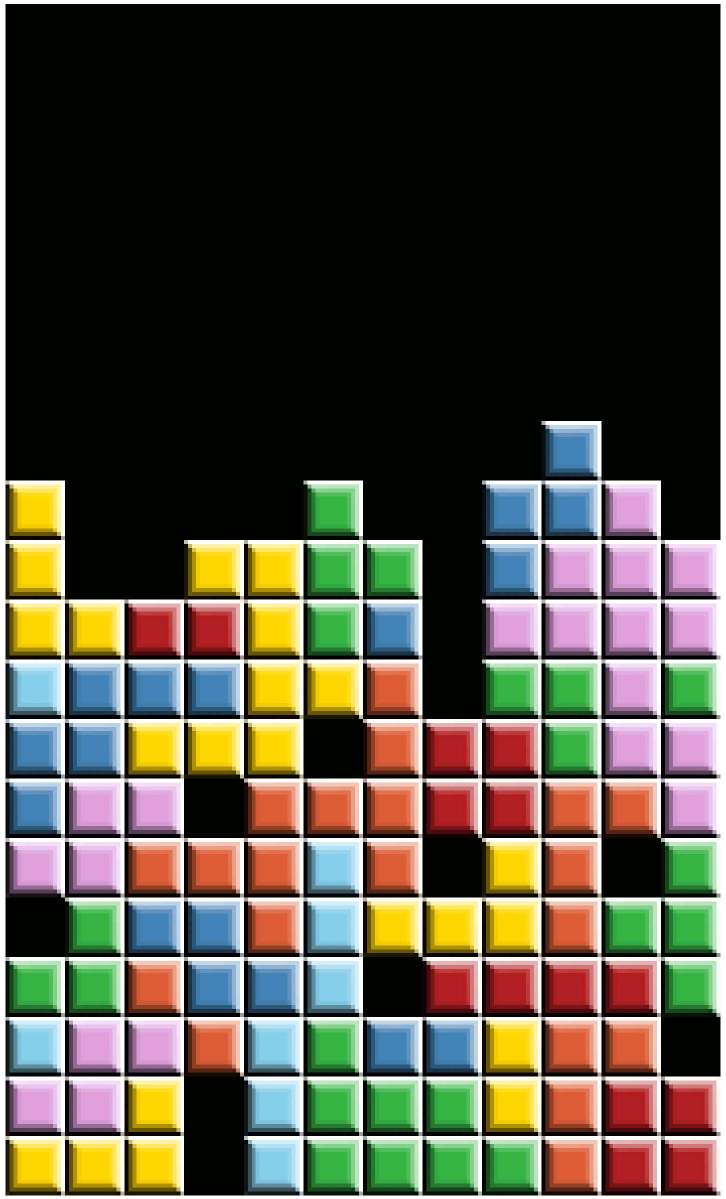


ENHANCING ARTIFICIAL INTELLIGENCE IN GAMING THROUGH QUANTUM COMPUTING: A TETRIS-THEMED COMPARATIVE ANALYSIS

By Jalen Packer

Mentor – Dr. Dvijesh Shastri

University of Houston Downtown | Fall 2024

AGENDA

INTRODUCTION

Since the dawn of gaming AI, we've relied on traditional computing approaches that often lead to predictable behavior patterns

This limitation sparked my investigation into how quantum computing might revolutionize gaming AI

This research intends to explore the potential of quantum computing in gaming using Tetris as my testing ground

OBJECTIVES

Explore Classic & Quantum Approaches	Research appropriate classic models and quantum algorithms
Implement Tetris-based Prototypes	Build a working *Heuristic* model able to play Tetris Build a working *QAOA* model able to play Tetris
Evaluate Performance	Determine metrics for comparison Measure and calculate model statistics
Generate Actionable Insights	Analyze performance metrics Make observations and assumptions Identify future solutions

LITERATURE REVIEW

I chose to build a **HEURISTIC** model as my classic-computed competitor.

- Focus on speed and efficiency over accuracy
- Uses rules of thumb and shortcuts
- Reduced Complexity: Heuristics lower the number of potential moves to evaluate

Why?

Simplicity and efficiency; heuristics are most common in modern games

LITERATURE REVIEW CONT.

I chose to build a **Quantum Approximate Optimization Algorithm (QAOA)** -based model as my quantum-computed competitor.

- For combinatorial optimization problems
- Apply alternating layers of:
  - Problem Hamiltonian (encodes the optimization problem)
  - Mixer Hamiltonian (explores solution space)
- Measures results by optimizing parameters and repeating this until an optimal solution is found

Why?

QAOA is a quantum algorithm claimed to be designed best for puzzle solving; suits a game like Tetris

TETRIS EXPLANATION

Game Overview

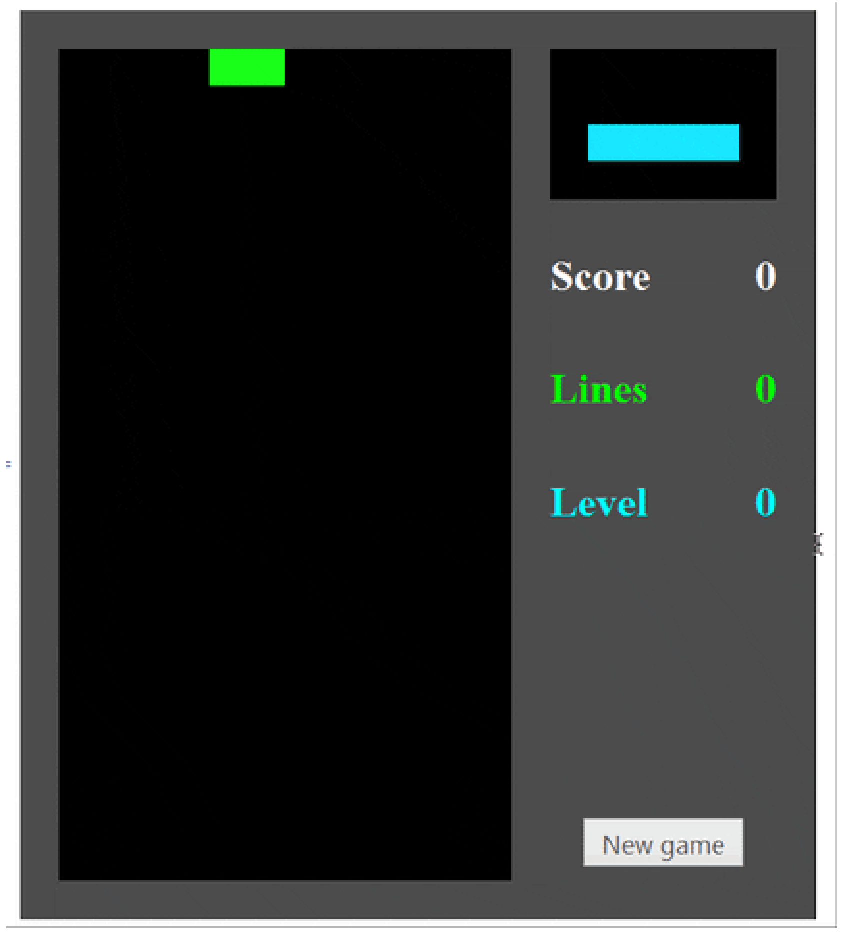
- Classic puzzle game where players manipulate falling geometric pieces ('tetrominoes')
- Seven unique piece shapes: I, J, L, O, S, T, and Z
- Pieces fall from top of a 10x20 grid playing field

Core Mechanics

- Rotate pieces (90° turns)
- Move pieces left/right
- Accelerate piece descent
- Perform "hard drops" (instant placement)

Scoring System

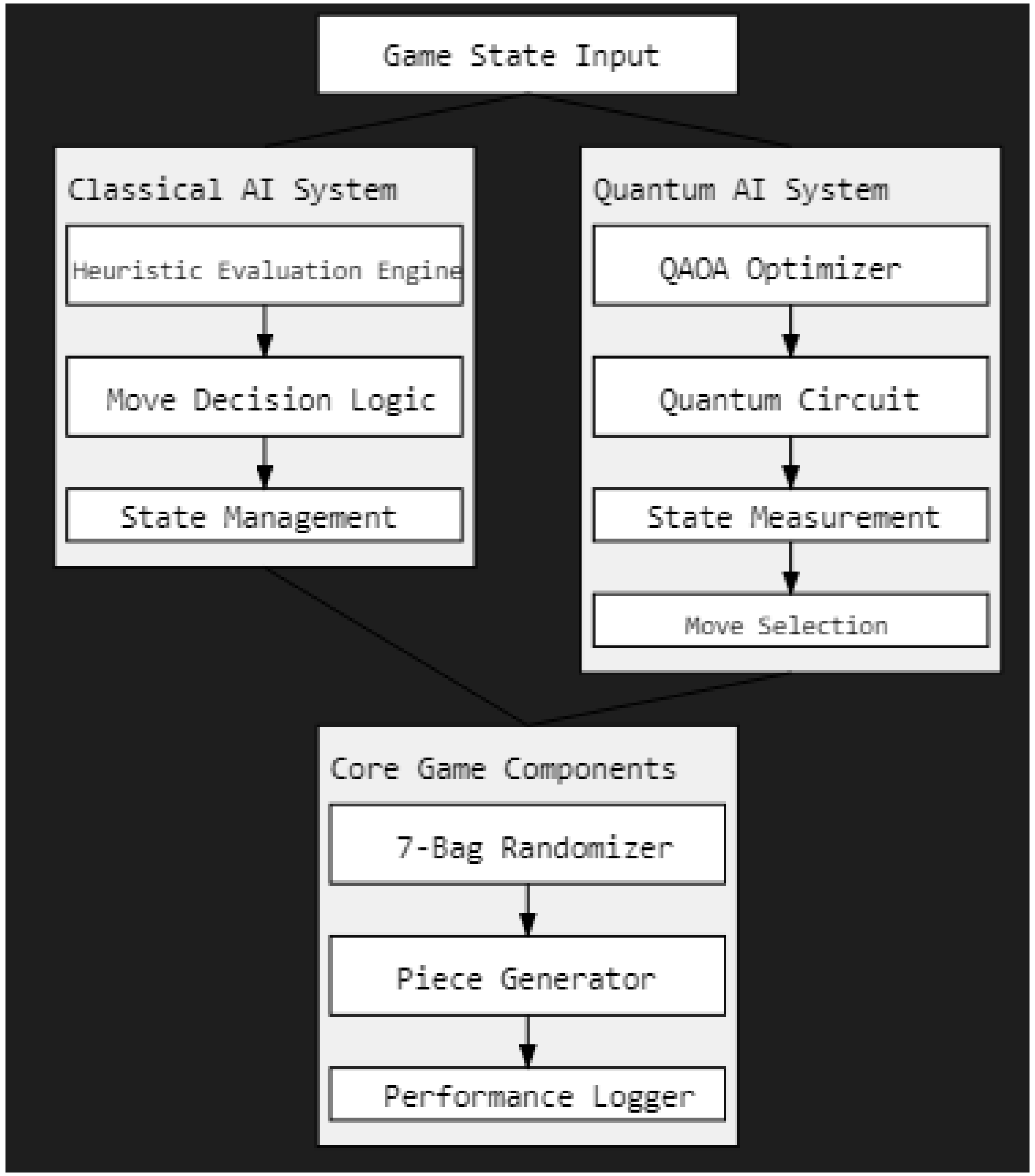
- Completing horizontal lines (40-1200 points)
- Multiple line clears (higher scores)



SYSTEM ARCHITECTURES

Development Environment & Components Overview

- Visual Studio Code IDE
- Python primary language
- Pygame for game environment
- Google Cirq
- Additional dependencies for QAOA model
- Minimized Tetris game testing ground



HEURISTIC MODEL IMPLEMENTATION

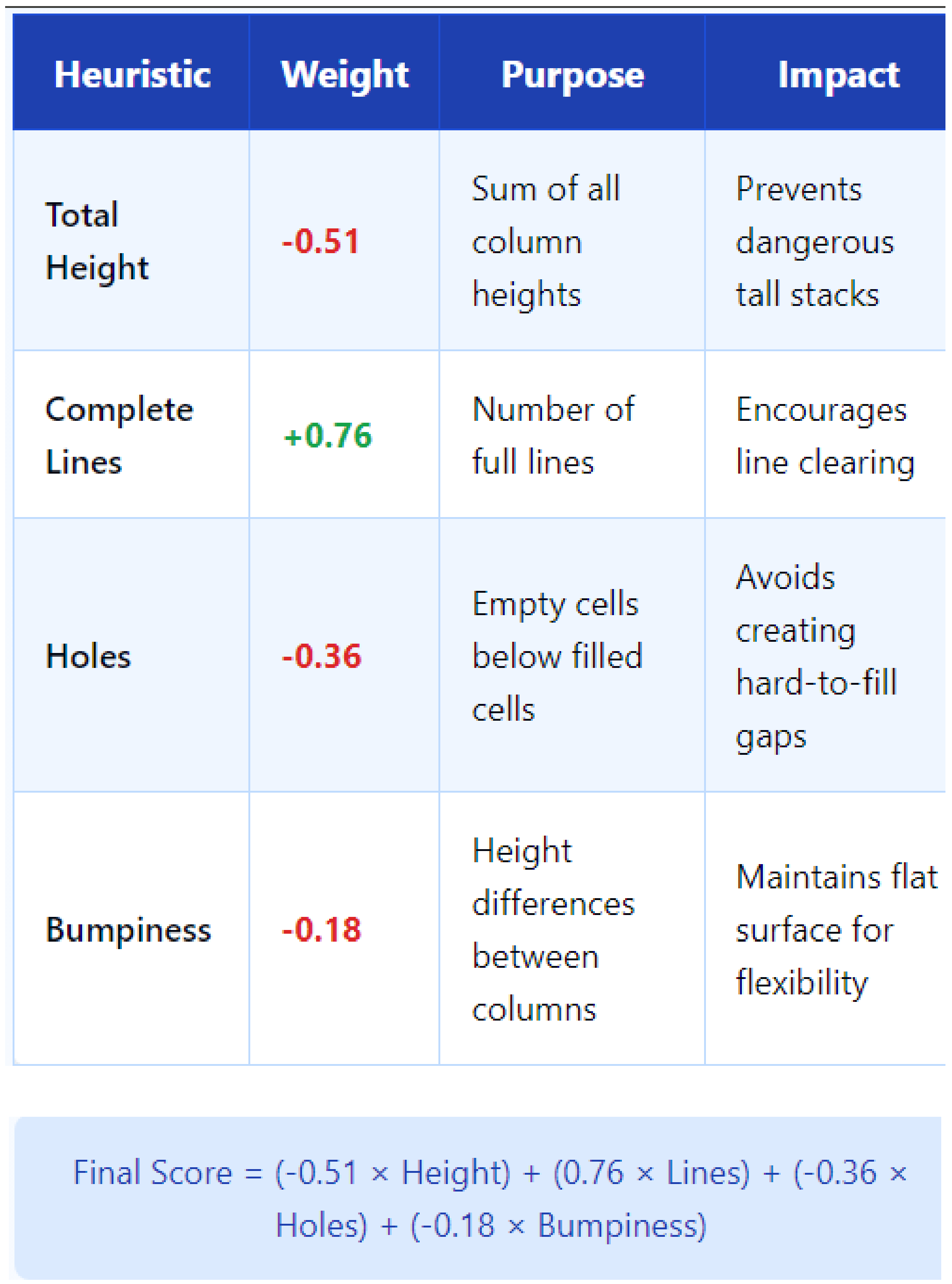
1. For each piece, it generates all possible positions and rotations

2. Evaluates each possibility using four weighted metrics:
   - Total Height (penalize tall stacks)
   - Complete Lines (reward clearable lines)
   - Holes (penalize empty gaps)
   - Bumpiness (penalize uneven surface)

   Weight values explained here:
   [13] Y. Lee, Apr. 2013
   Tetris AI – The (Near) Perfect Bot | Code My Road

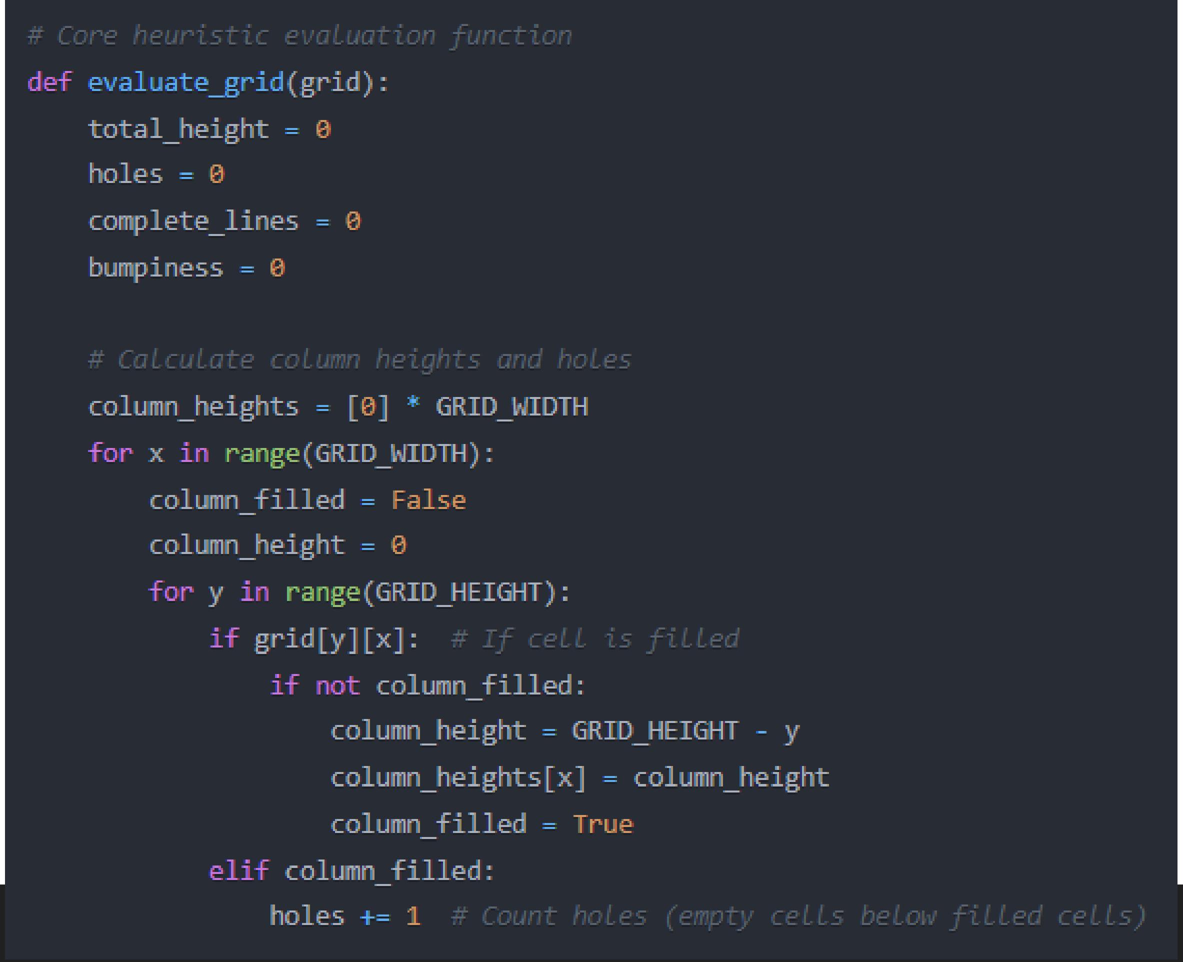
3. Chooses and executes the move with highest score

4. Repeats process for next piece



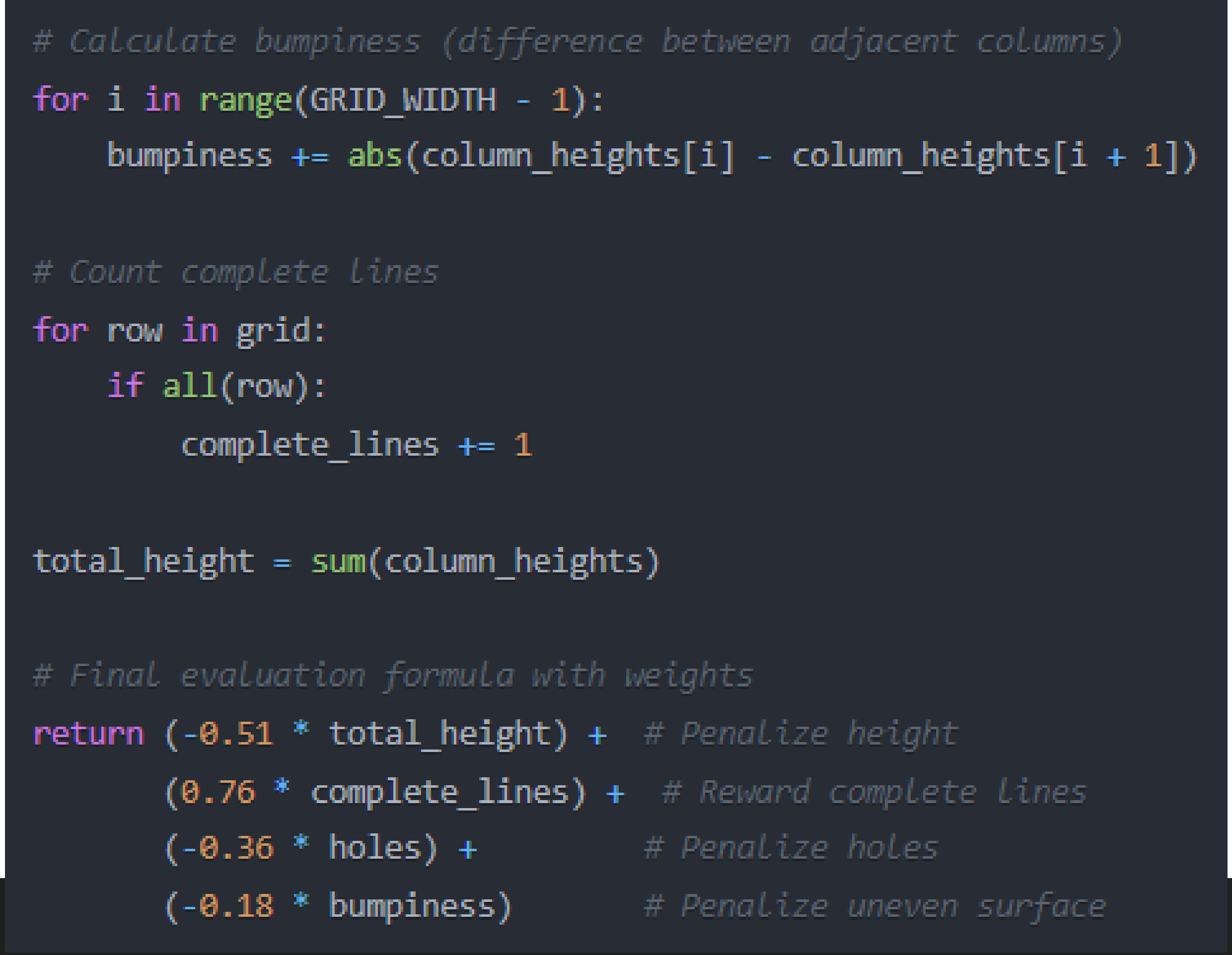
Heuristic	Weight	Purpose	Impact
Total Height	-0.51	Sum of all column heights	Prevents dangerous tall stacks
Complete Lines	+0.76	Number of full lines	Encourages line clearing
Holes	-0.36	Empty cells below filled cells	Avoids creating hard-to-fill gaps
Bumpiness	-0.18	Height differences between columns	Maintains flat surface for flexibility

Final Score = (-0.51 × Height) + (0.76 × Lines) + (-0.36 × Holes) + (-0.18 × Bumpiness)



```python
# Core heuristic evaluation function
def evaluate_grid(grid):
    total_height = 0
    holes = 0
    complete_lines = 0
    bumpiness = 0

    # Calculate column heights and holes
    column_heights = [0] * GRID_WIDTH
    for x in range(GRID_WIDTH):
        column_filled = False
        column_height = 0
        for y in range(GRID_HEIGHT):
            if grid[y][x]:  # If cell is filled
                if not column_filled:
                    column_height = GRID_HEIGHT - y
                    column_heights[x] = column_height
                    column_filled = True
            elif column_filled:
                holes += 1  # Count holes (empty cells below filled cells)
```



```python
# Calculate bumpiness (difference between adjacent columns)
for i in range(GRID_WIDTH - 1):
    bumpiness += abs(column_heights[i] - column_heights[i + 1])

# Count complete lines
for row in grid:
    if all(row):
        complete_lines += 1

total_height = sum(column_heights)

# Final evaluation formula with weights
return (-0.51 * total_height) +  # Penalize height
       (0.76 * complete_lines) +  # Reward complete lines
       (-0.36 * holes) +          # Penalize holes
       (-0.18 * bumpiness)        # Penalize uneven surface
```

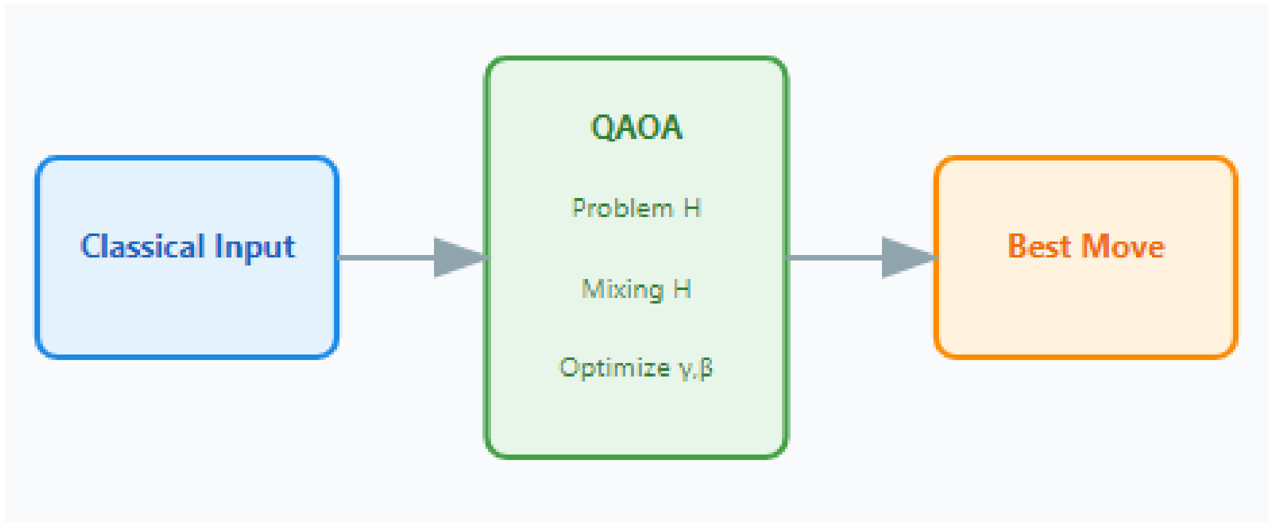
HEURISTIC MODEL CODE

QAOA MODEL IMPLEMENTATION

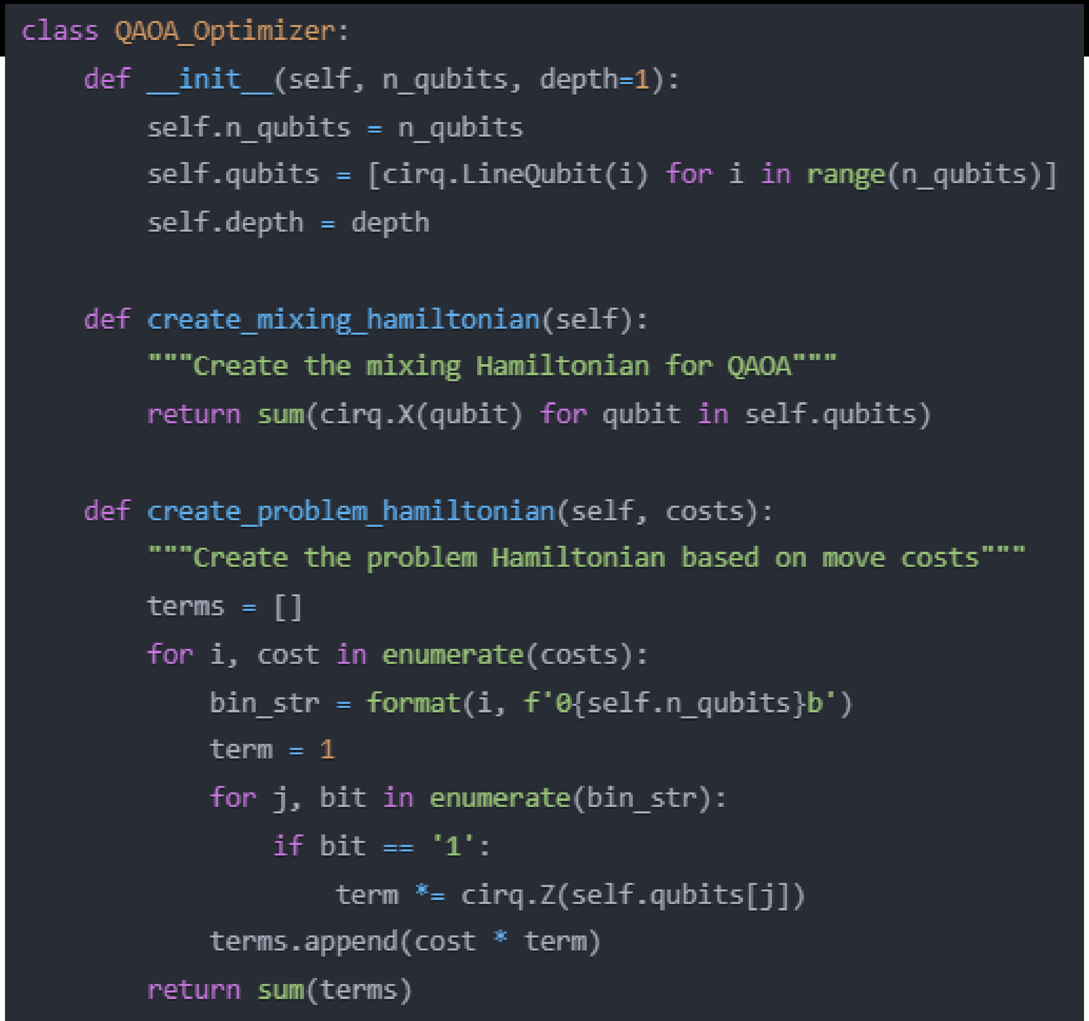
1. **QAOA Integration:**
   - Uses Cirq for quantum circuit simulation
   - Implements full QAOA optimization process

2. **Main Quantum Components:**
   - *Problem Hamiltonian*: Encodes move scores into quantum operations
   - *Mixing Hamiltonian*: Enables quantum exploration of move space
   - *QAOA Circuit*: Combines problem and mixing operations
   - *Parameter Optimization*: Finds optimal γ and β values

3. **Quantum Approach:**
   - *Classical part*: Generates possible moves
   - *Quantum part*: Uses QAOA to optimize move selection
   - *Integration*: Combines both to make final move decisions



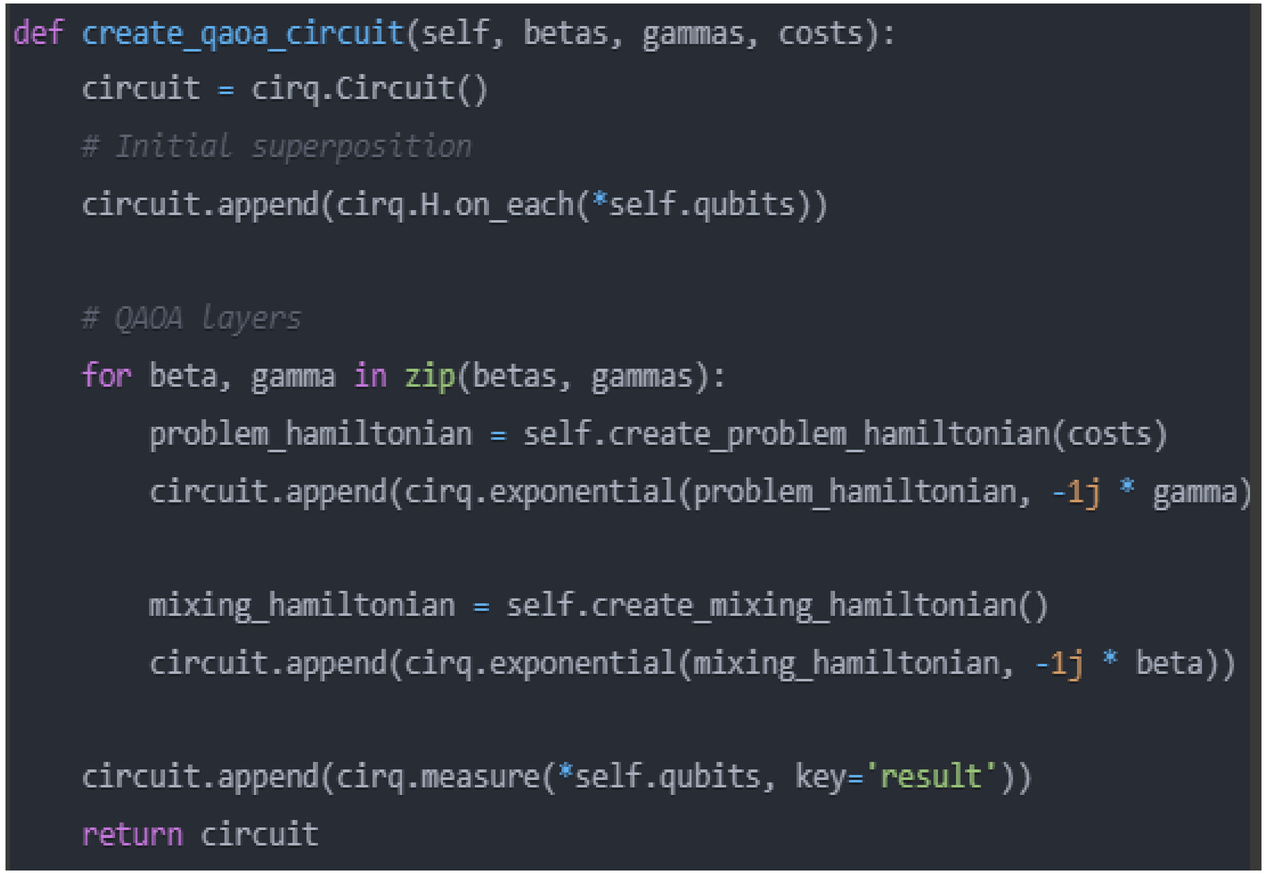
QAOA MODEL CODE



```python
class QAOA_Optimizer:
    def __init__(self, n_qubits, depth=1):
        self.n_qubits = n_qubits
        self.qubits = [cirq.LineQubit(i) for i in range(n_qubits)]
        self.depth = depth

    def create_mixing_hamiltonian(self):
        """Create the mixing Hamiltonian for QAOA"""
        return sum(cirq.X(qubit) for qubit in self.qubits)

    def create_problem_hamiltonian(self, costs):
        """Create the problem Hamiltonian based on move costs"""
        terms = []
        for i, cost in enumerate(costs):
            bin_str = format(i, f'0{self.n_qubits}b')
            term = 1
            for j, bit in enumerate(bin_str):
                if bit == '1':
                    term *= cirq.Z(self.qubits[j])
            terms.append(cost * term)
        return sum(terms)
```



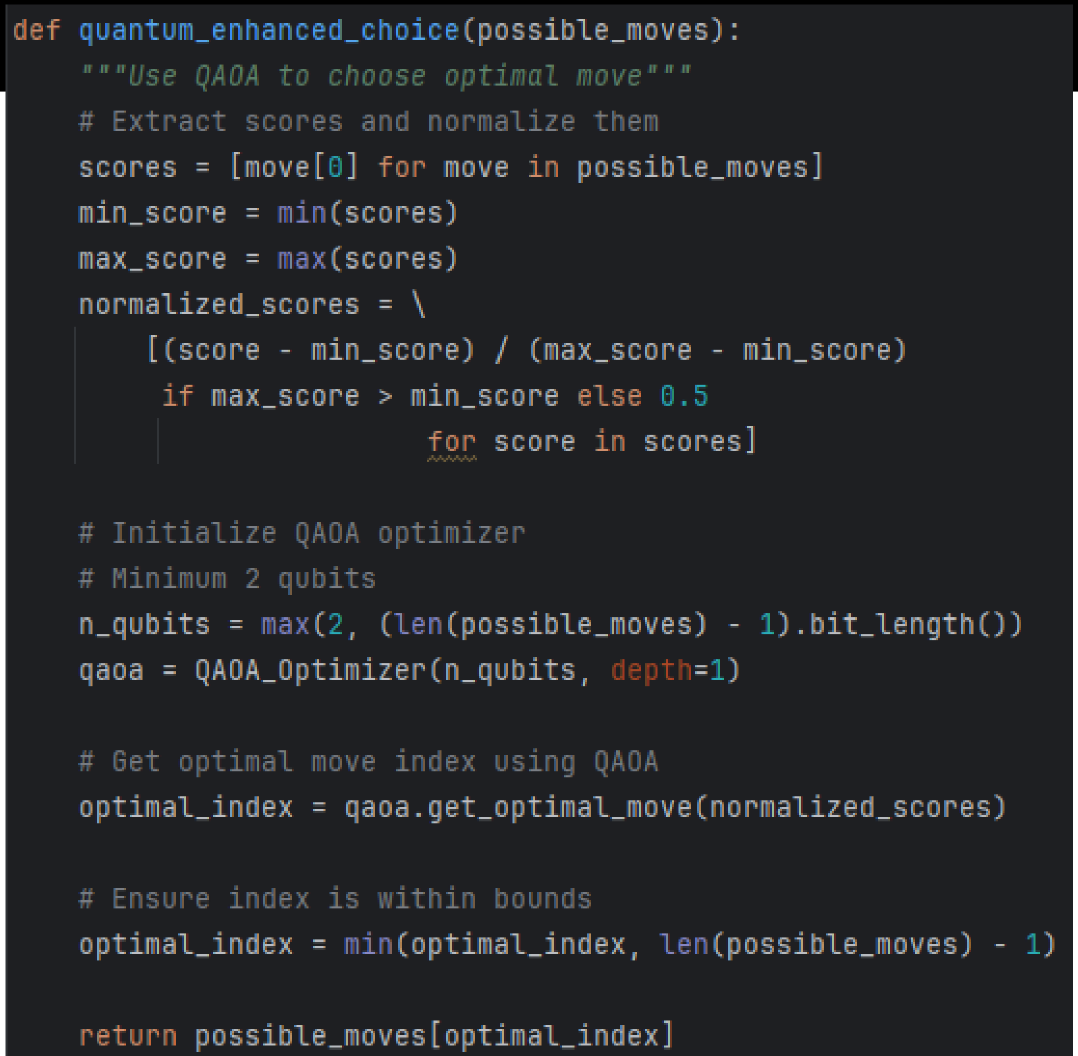
```python
def create_qaoa_circuit(self, betas, gammas, costs):
    circuit = cirq.Circuit()
    # Initial superposition
    circuit.append(cirq.H.on_each(*self.qubits))

    # QAOA layers
    for beta, gamma in zip(betas, gammas):
        problem_hamiltonian = self.create_problem_hamiltonian(costs)
        circuit.append(cirq.exponential(problem_hamiltonian, -1j * gamma)

        mixing_hamiltonian = self.create_mixing_hamiltonian()
        circuit.append(cirq.exponential(mixing_hamiltonian, -1j * beta))

    circuit.append(cirq.measure(*self.qubits, key='result'))
    return circuit
```

QAOA MODEL CODE CONT.



```python
def quantum_enhanced_choice(possible_moves):
    """Use QAOA to choose optimal move"""
    # Extract scores and normalize them
    scores = [move[0] for move in possible_moves]
    min_score = min(scores)
    max_score = max(scores)
    normalized_scores = \
        [(score - min_score) / (max_score - min_score)
         if max_score > min_score else 0.5
                         for score in scores]

    # Initialize QAOA optimizer
    # Minimum 2 qubits
    n_qubits = max(2, (len(possible_moves) - 1).bit_length())
    qaoa = QAOA_Optimizer(n_qubits, depth=1)

    # Get optimal move index using QAOA
    optimal_index = qaoa.get_optimal_move(normalized_scores)

    # Ensure index is within bounds
    optimal_index = min(optimal_index, len(possible_moves) - 1)

    return possible_moves[optimal_index]
```

PERFORMANCE ANALYSIS

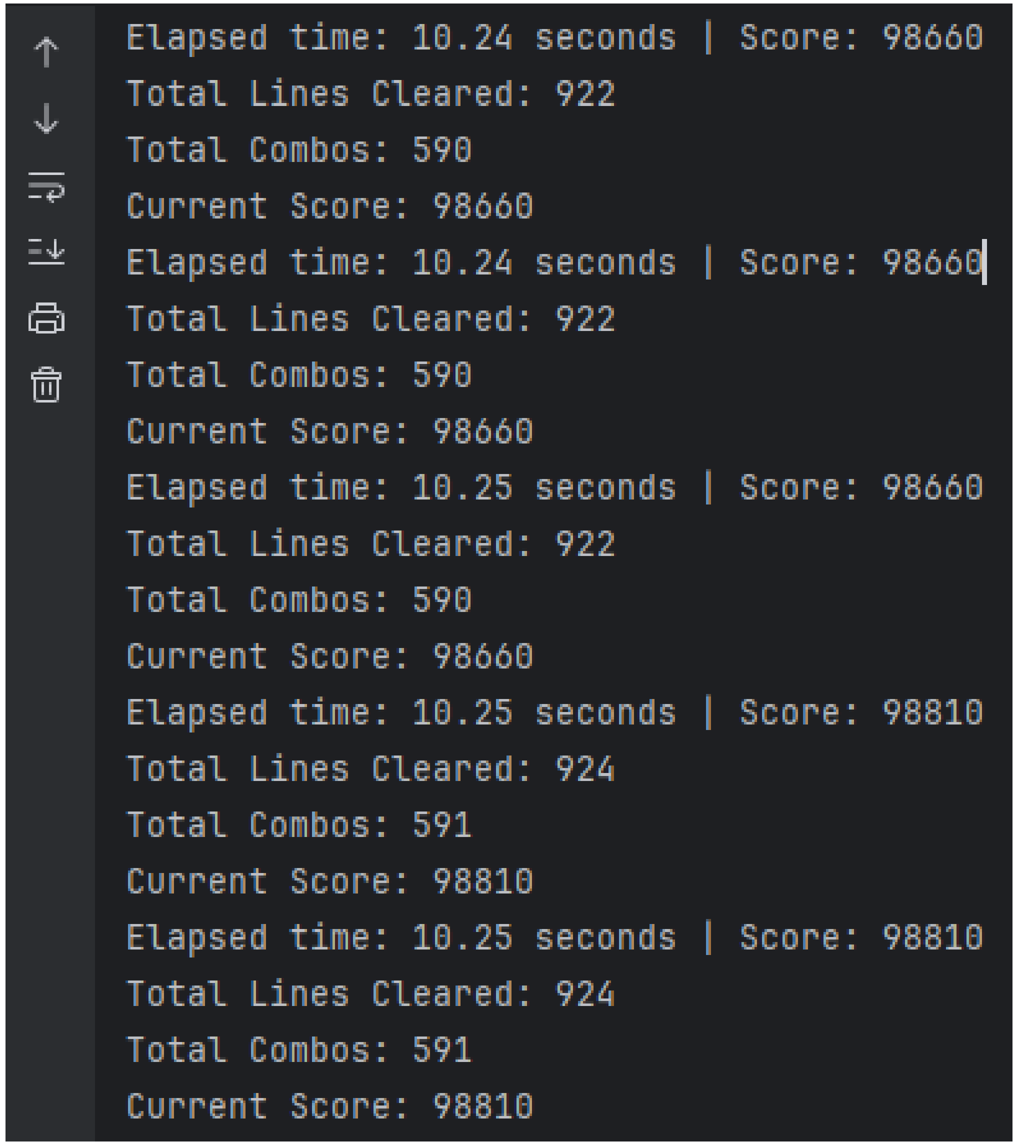
Evaluation Metrics

- Elapsed time until failure
- Score
- Number of Lines Cleared
- Number of Combos

Analysis Methods

- 3-round testing with 10 runs each
- Statistical validation

Comparative Analysis Document Link:
https://docs.google.com/document/d/11bnzQbQgXNEOI5qxBFCvvxvj2NbFECE5I7Nxg42SYms/edit?usp=sharing



```
Elapsed time: 10.24 seconds | Score: 98660
Total Lines Cleared: 922
Total Combos: 590
Current Score: 98660
Elapsed time: 10.24 seconds | Score: 98660
Total Lines Cleared: 922
Total Combos: 590
Current Score: 98660
Elapsed time: 10.25 seconds | Score: 98660
Total Lines Cleared: 922
Total Combos: 590
Current Score: 98660
Elapsed time: 10.25 seconds | Score: 98810
Total Lines Cleared: 924
Total Combos: 591
Current Score: 98810
Elapsed time: 10.25 seconds | Score: 98810
Total Lines Cleared: 924
Total Combos: 591
Current Score: 98810
```

VISUALIZATION

Heuristic Model Demo Link:
https://drive.google.com/file/d/1afc2SpINrN9uBNpFqoyMIHcONP11joHW/view?usp=drive

QAOA Model Demo Link:
https://drive.google.com/file/d/1dMLNnMebTxetOxSiy7rTS1YspTLkK3KK/view?usp=drive_link

COMPARATIVE RESULTS

Key Findings

- 6.01% higher scoring efficiency in quantum model
- Quantum: 150MB memory, 10ms decisions
- Classical: 15MB memory, 15ms decisions
- The heuristic model had the lowest floors and the highest ceilings, implying its higher dexterity
- The heuristic model was more consistent overall
- The QAOA model scored above average more consistently

Table 1: Scoring Efficiency Across Rounds

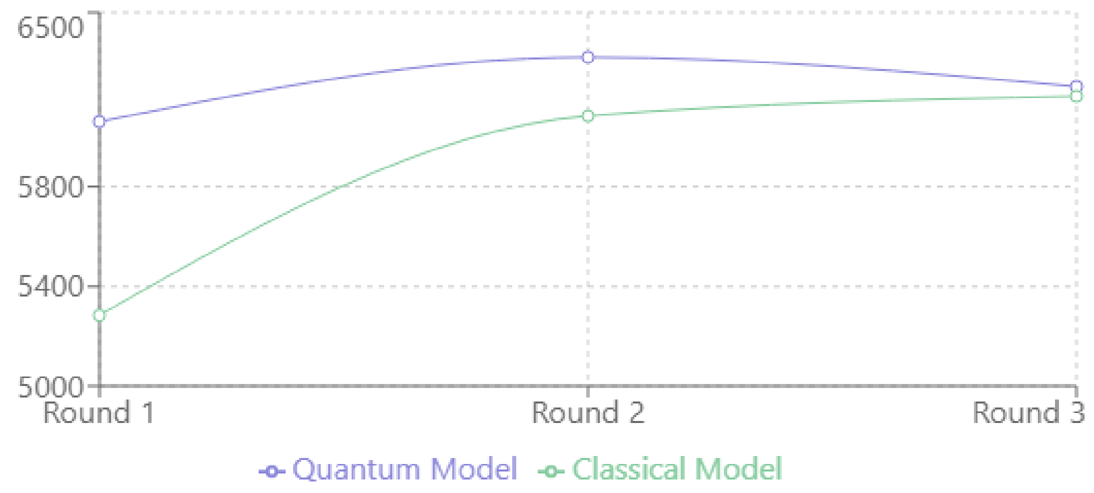
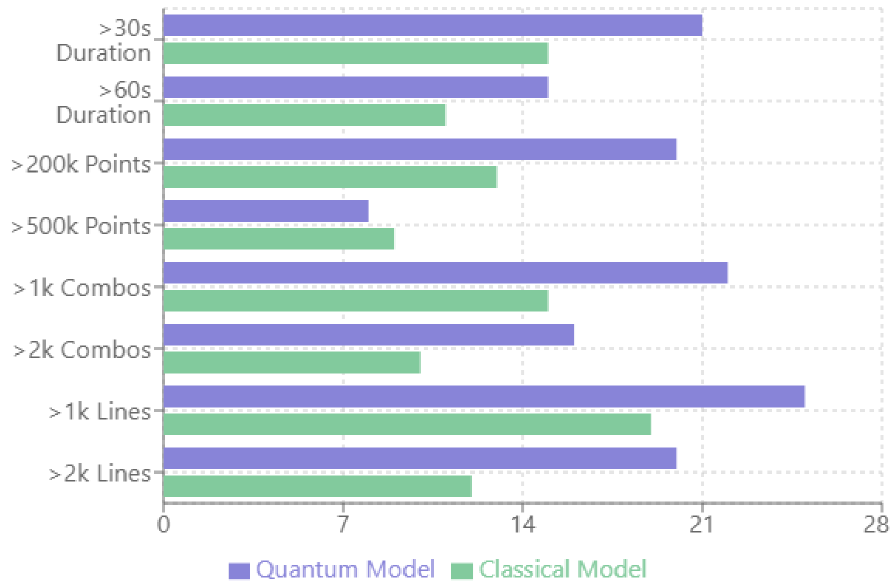


Table 2: Performance Thresholds Comparison



CHALLENGES ENCOUNTERED

Challenges

- Quantum framework issues
- QAOA Implementation complexity
- Time

Solutions

- Experimenting with new frameworks
- Documentation Research
- Optimization strategies and Performance tuning

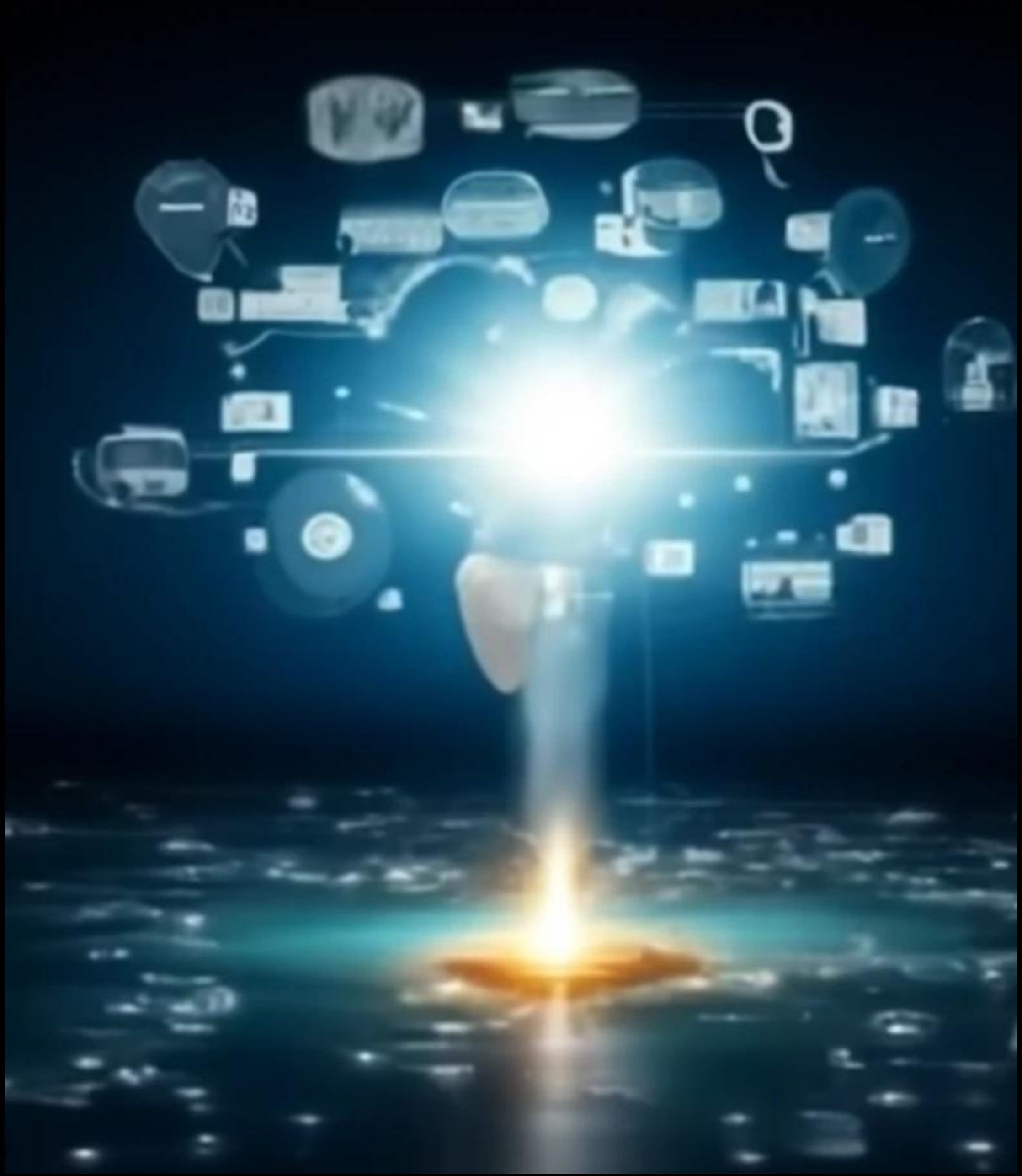
FUTURE WORK

Development Opportunities

- Hybrid system optimization
- Enhanced quantum algorithms
- Broader gaming applications

Research Directions

- Real-time optimization
- Resource efficiency
- Scalability solutions



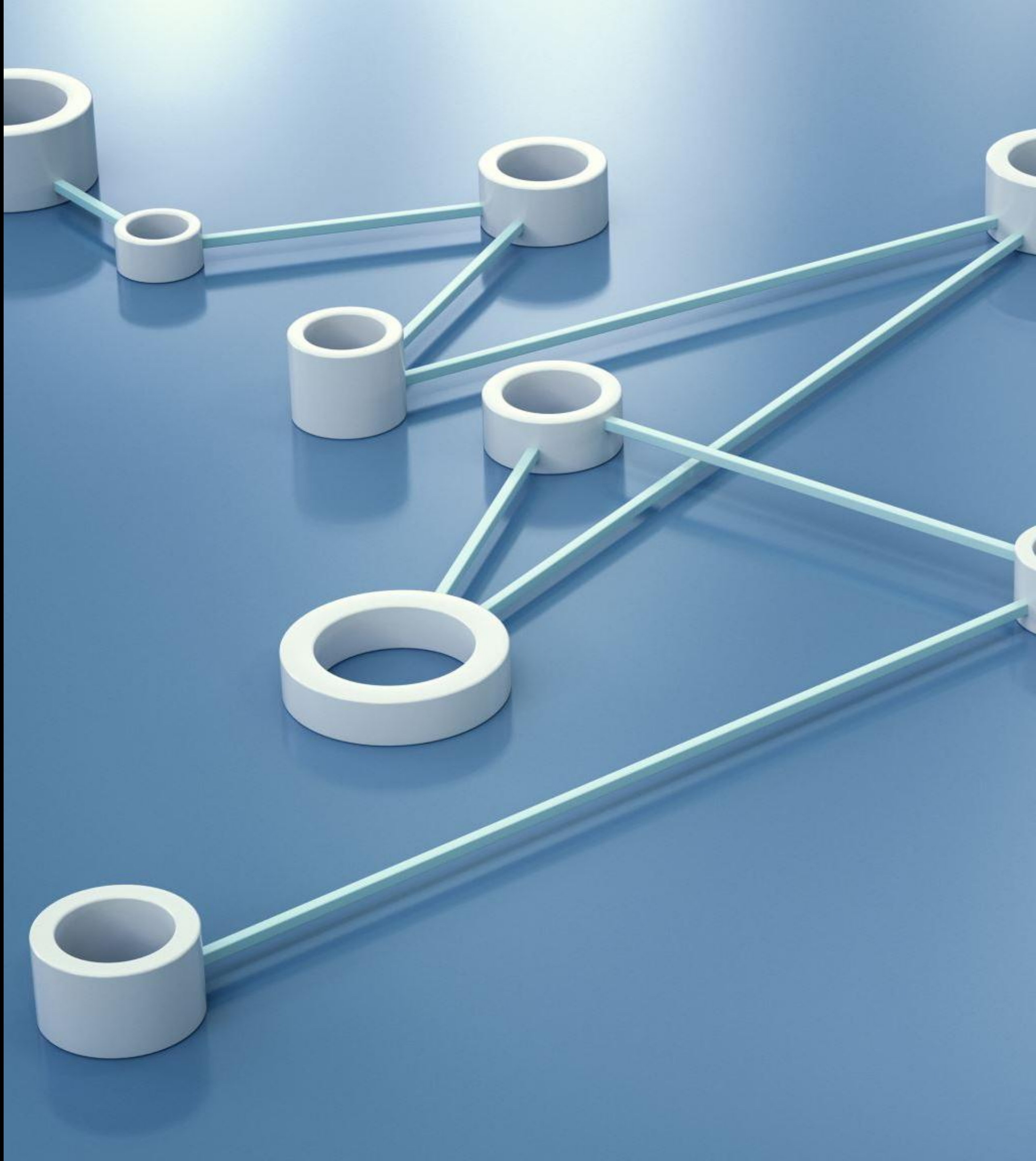
CONCLUSION

Key Achievements

- Successful quantum implementation
- Performance improvements
- Practical insights gained

Impact

- Gaming advancement
- Quantum computing applications
- Future development paths



REFERENCES

1. H. Abraham et al., "Qiskit: An Open-source Framework for Quantum Computing," *Zenodo*, Jan. 2019, doi: 10.5281/zenodo.2562110.
2. S. Adebayo, "How our inventions beat us at our own games: AI game strategies," *Deepgram*, Jan. 2024. [Online]. Available: https://www.deepgram.com/blog/ai-game-strategies
3. O. Ayoade, P. Rivas, and J. Orduz, "Artificial intelligence computing at the quantum level," *Data*, vol. 7, no. 3, pp. 28-42, Mar. 2022, doi: 10.3390/data7030028.
4. K. Becker, "Flying unicorn: Developing a game for a quantum computer," *arXiv*, Oct. 2019. [Online]. Available: https://arxiv.org/abs/1910.08238
5. Boris Inc., "Classic Tetris Gameplay Example," *GIPHY*, accessed Nov. 24, 2024. [Online]. Available: https://media.giphy.com/v1/xT0Gqcmc4NsPgBVQEU/giphy.gif
6. F. Bova, A. Goldfarb, and R. G. Melko, "Commercial applications of quantum computing," *EPJ Quantum Technology*, vol. 8, no. 1, pp. 2-14, Jan. 2021, doi: 10.1140/epjqt/s40507-021-00091-1.
7. Carpe Flux Capacitor, "Craiyon AI Image Generator," *Craiyon*, 2024. [Online]. Available: https://www.craiyon.com/
8. Cirq Developers, "Cirq Documentation," *Google Quantum AI*, Dec. 2023. [Online]. Available: https://quantumai.google/cirq
9. J. Du et al., "Experimental realization of quantum games on a quantum computer," *Phys. Rev. Lett*., vol. 88, no. 13, Art. no. 137902, Apr. 2002, doi: 10.1103/PhysRevLett.88.137902.
10. E. Farhi, J. Goldstone, and S. Gutmann, "A Quantum Approximate Optimization Algorithm," *arXiv*, Nov. 2014. [Online]. Available: https://arxiv.org/abs/1411.4028

REFERENCES CONT.

11. GIPHY Studios, "Tetris AI Gameplay Demonstration," *GIPHY*, accessed Nov. 24, 2024. [Online]. Available: https://media.giphy.com/v1/MOSebUr4rvZS0/giphy.gif

12. F. S. Khan and S. J. Phoenix, "Gaming the quantum," *arXiv*, Feb. 2012. [Online]. Available: https://arxiv.org/abs/1202.1142

13. Y. Lee, "Tetris AI: The near perfect player," *Code My Road*, Apr. 2013, Available: https://codemyroad.wordpress.com/2013/04/14/tetris-ai-the-near-perfect-player/

14. Y. Lu and W. Li, "Techniques and paradigms in modern game AI systems," *Algorithms*, vol. 15, no. 8, pp. 282-301, Aug. 2022, doi: 10.3390/a15080282.

15. A. Montanaro, "Quantum algorithms: an overview," *npj Quantum Information*, vol. 2, no. 1, Art. no. 15023, Jan. 2016, doi: 10.1038/npjqi.2015.23.

16. M. A. Nielsen and I. L. Chuang, Quantum Computation and Quantum Information: 10th Anniversary Edition. *Cambridge, UK: Cambridge University Press*, 2010.

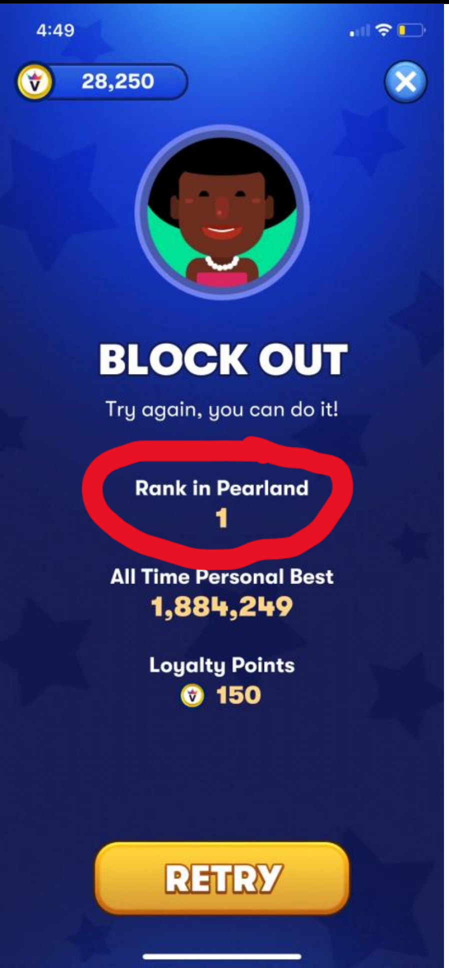
17. R. Prevedel, A. Stefanov, P. Walther, and A. Zeilinger, "Experimental realization of a quantum game on a one-way quantum computer," *New J. Phys*., vol. 9, no. 6, Art. no. 205, Jun. 2007, doi: 10.1088/1367-2630/9/6/205.

18. N. Skult and J. Smed, "The marriage of quantum computing and interactive storytelling," in *Games and Narrative: Theory and Practice, Cham*, Switzerland: Springer, 2021, pp. 191-206.

19. S. Srivastava, "How AI in gaming is propelling the industry into a new epoch," *AppInventiv*, Jan. 2024. [Online]. Available: https://appinventiv.com/blog/ai-in-gaming/

20. Tech With Tim, "How to create an unbeatable Tetris AI," *YouTube*, Nov. 2018. [Online]. Available: https://youtu.be/uoR4ilCWwKA

Q&A


4:49

28,250

BLOCK OUT

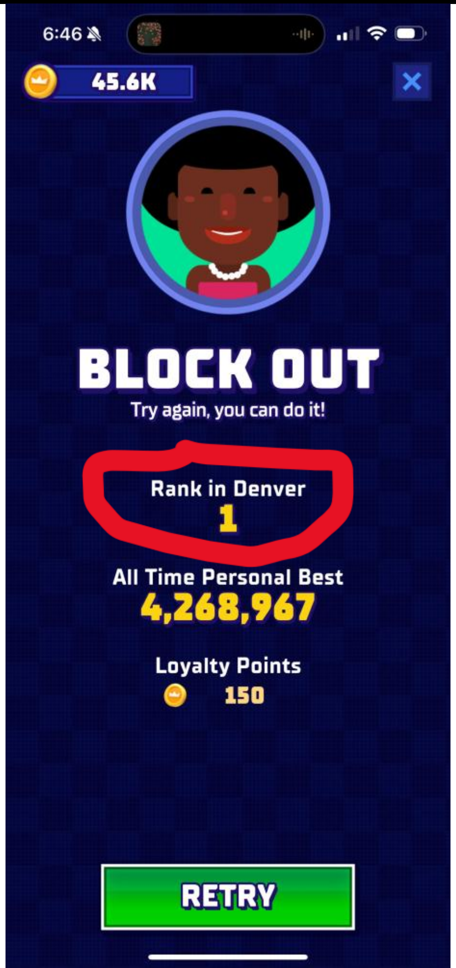
Try again, you can do it!

Rank in Pearland
1

All Time Personal Best
1,884,249

Loyalty Points
150

RETRY


6:46

45.6K

BLOCK OUT

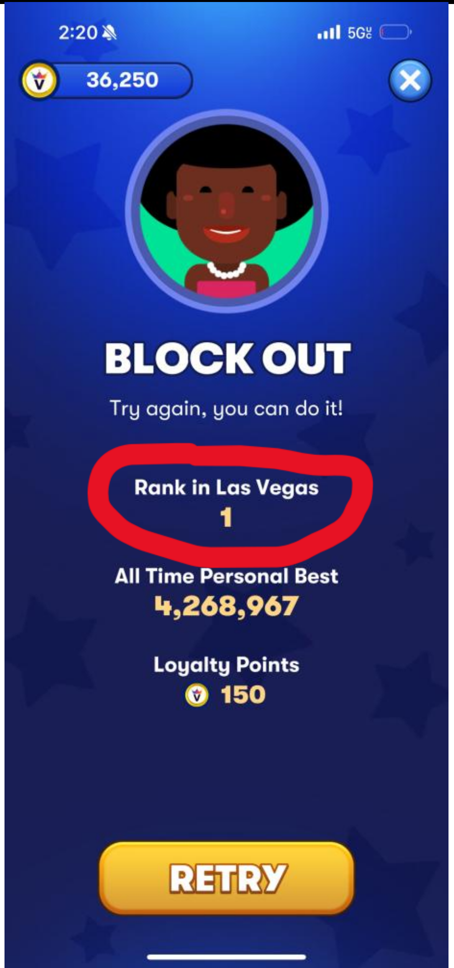
Try again, you can do it!

Rank in Denver
1

All Time Personal Best
4,268,967

Loyalty Points
150

RETRY


2:20

36,250

BLOCK OUT

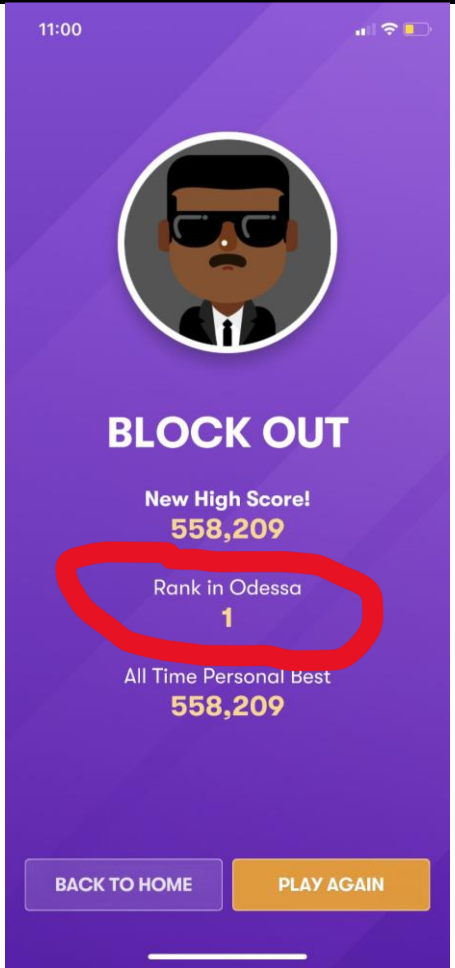
Try again, you can do it!

Rank in Las Vegas
1

All Time Personal Best
4,268,967

Loyalty Points
150

RETRY


11:00

BLOCK OUT

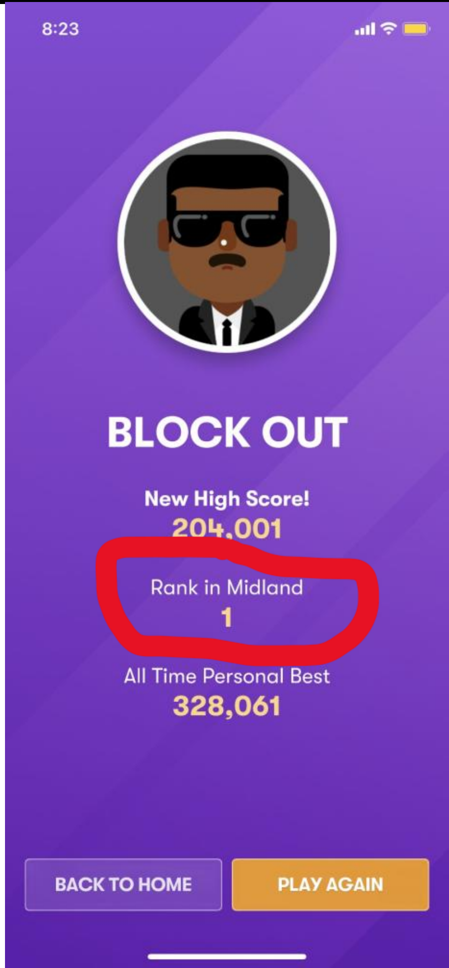
New High Score!
558,209

Rank in Odessa
1

All Time Personal Best
558,209

BACK TO HOME

PLAY AGAIN


8:23

BLOCK OUT

New High Score!
204,001

Rank in Midland
1

All Time Personal Best
328,061

BACK TO HOME

PLAY AGAIN