Chapter 1

Testing

1.1 Iterative Testing

I have been playtesting the program throughout the development process to find any bugs and fix them accordingly. However, a few issues have required additional measures.

1.1.1 Minimax

Since minimax is recursive algorithm, debugging it has proven a challenge. I have therefore configured the Python `logging` library to help collect information on the minimax tree for every function call.

`base.py`

```python
    def print_stats(self, score, move):
        """
        Prints statistics after traversing tree.

        Args:
            score (int): Final score obtained after traversal.
            move (Move): Best move obtained after traversal.
        """
        if self._verbose is False:
            return

        self._stats['time_taken'] = round(1000 * (time.time() - self._stats['time_taken']), 3)
        self._stats['ms_per_node'] = round(self._stats['time_taken'] / self._stats['nodes'], 3)

        # Prints stats across multiple lines
        if self._verbose is True:
            logger.info(f'{self.__str__()} Search Results:')
            logger.info(printer.pformat(self._stats))
            logger.info(f'Best score:  {score}   Best move: {move}')

        # Prints stats in a compacted format
        elif self._verbose.lower() == 'compact':
            logger.info(self._stats)
            logger.info(f'Best score: {score}   Best move: {move}')
```

Listing 1.1: BaseCPU Method for logging minimax statistics

1.1.2 Migrations

To correct errors made to the `games` table, since recreating it would mean deleting all existing games, I have opted to use migrations to fix bugs by editing the table schema, as shown in Section ??.

1.2 Unit Tests

1.2.1 Board Evaluator

To test every aspect of the evaluation function, I have set up some unit tests with custom positions using my editor screen. These positions are designed to test every aspect of the evaluation, along with some obviously imbalanced positions to test the overall accuracy of the evaluation function. All positions are set up to give an advantage to the blue player.

Evaluating	**FEN string**	**Score**	**Passed**
Material	`sc9/10/10/4paPa4/5Pa4/10/10/9Sa b`	124	✓
Position	`sc9/4nanana3/10/10/10/4NaNaNa3/10/9Sa b`	66	✓
Mobility	See footnote[1]	196	✓
King Safety	`sc4fa3pa/10/10/10/10/10/5FaPa2Sa b`	3	✓
Combined	See footnote[2]	437	✓

Table 1.1: Board evaluator test results

1.2.2 CPU

Similarly, to evaluate the strength of my CPU, I have setup some custom positions that I already know the best continuation of, and run each CPU engine on them to test if they can solve it.

sc9/pafa8/Fa9/10/10/10/10/9Sa b

I have also personally played against the CPU engines to gauge its strength in a realistic setting. The results are shown below:

Me	**CPU**
2	5

Table 1.2: Score of me vs CPU (I am very bad)

Mr Myslov	**CPU**
1	6

Table 1.3: Score of Mr Myslov vs CPU (Mr Myslov is even worse)

[1] scpapa7/papapa7/papapa1Pa1Pa1Pa1/10/4Pa1Pa1Pa1/10/4Pa1Pa3/9Sa b

[2] scnc1fcncpbpb3/pa9/pb1pc1rbpa3Pd/1Pc2Pd4Pc/2Pd1RaRb4/10/7Pa2/2PdNaFaNa3Sa b

1.2.3 Shadow Mapping

To test the shadow mapping algorithm, I have set up some occluding objects togehter with a light source. Since visuals are subjective, me and my client have deemed the following results to be adequate.

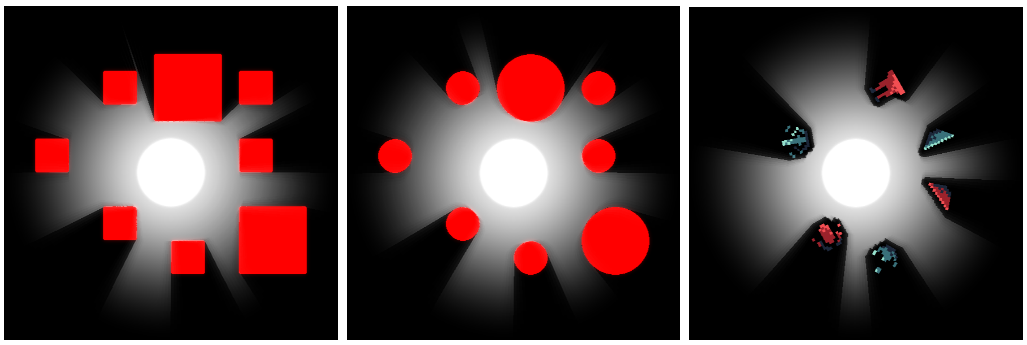

Figure 1.1: Shadow mapping algorithm tests (softShadow=0.5, radius=0.5)

1.3 Final Tests

1.3.1 Objective 1

1.3.2 Objective 2

1.3.3 Objective 3

1.4 Videos