Mini-Nea

Brandon Francis

April 2025

1 Analysis

1.1 The Problem

Users of the program wish to be able to simulate the Angel Problem. Which, since it was first invented in 1982 by John Conway, has not had many 2-player simulators. The problem involves 2 players who assume the role of an Angel and the Devil. They act on an infinite grid where, once per turn, the Devil can place one block anywhere on the grid, then the angel can move a number of squares in any direction up to its power; however, they are unable to move onto a space that the Devil has blocked. The goal of the Devil is to block and trap the Angel in order to win. The simulation will represent how an angel of any power can operate and try to survive against a devil. The game will end either when the angel is trapped or when the users stop the program.

1.2 Target User

1.3 Who are they?

The intended users for this project are teachers who want to demonstrate the complexities of uneven games, such as the angel problem, to their students. To gather information on what possible design requirements may be needed. For this I interviewed a computer science teacher who wanted to model the angel problem for their students. The following sections detail the questions that were asked of that teacher.

1.3.1 General Questions

the question. Q — What specific aspects of the angel problem are you most interested in exploring or understanding better through this tool?

> A — I would like to be able to demonstrate the angel problem, including different strengths, to classes. I would like students to be able to experiment with and against some basic strategies.

the question. Q — Are you more focused on simulating known strategies (like Kloster's or Máthé's), or on experimenting with new, user-created ones?

> A — Initially, the freedom to try different strategies would be good, but it would be an extension to include automated strategies. I am imagining 2 players at the same computer , or a teacher demonstrating both sides, or a single player experimenting.

the question. Q — Do you want the tool to emphasise visual intuition, formal strategy analysis, or both?

> A — I am not sure these are contradictory in requirements?

1.3.2 Gameplay questions

the question. Q — Would you prefer the angel's decisions to be automated (based on known strategies) or controlled manually by the user?

> A — This would be an extension option

the question. Q — Should the devil's moves also be automated, or should users be able to play as the devil?

> A — not automated, various automated devils could be an extension

the question. Q — would a 2-player version be useful?

> A — Yes (hotseat) would be the standard version; What should be really clear is whether it is the devil's or angel's turn

the question. Q — Would you like the ability to choose whether the angel always plays optimally or has restrictions (i.e only north/must move exactly k moves)?

> A — This would be a good extension option

the question. Q — Do you want to simulate specific scenarios (like the 1-angel, 2-angel cases), or should any angel power be selectable?

> A — Choosing the power of the angel at the start of the game is an essential component

the question. Q — Would you prefer turn-based controls with a "next move" button, or continuous simulation that you can pause and rewind?

> A — Turn based

the question. Q — How useful would it be for the simulation to include an undo/redo system?

A — This could be useful

the question. Q — Would it be useful to see a history or timeline of events (e.g a log of all the angel and devil moves)?

A — Not essential

the question. Q — Do you want the ability to edit the board manually, placing or removing blocks to create custom scenarios, or are you mainly interested in the general infinite grid?

A — Grid should be functionally infinite. You should be able to edit the board manually as the standard option.

the question. Q — How much mathematical assistance do you want? Would you rather the program gave you no help at all?

A — Not required, I think

the question. Q — Can the devil place a block on the square that the angel is on?

A — No

the question. Q — What controls would you want to use for the game?

A — I think that clicking on the square the devil wants to place a block on makes sense. Possibly a button to allow the devil to jump to a location further out. The angel could move by clicking or WASD?

the question. Q — For k¿=2, how would you like to choose how long to run the simulator for, input the number of steps or keep going until stop?

A — It should run until the devil wins or the player chooses to end.

1.3.3 Interface preferences

the question. Q — What would an ideal visual representation of the infinite grid look like to you? (e.g scrollable, zoomable, tiling?)

A — scrollable or jumping screen to screen is ok. There should be clear indicators of where you are relative to the starting location and

the angel's current location. This could be achieved with a coordinate system, and or mini-map, arrows/distances at the edge of the screen. The angel should have some indication of where/how far away the last devil block was placed.

the question. Q — Would a colour-coded or symbolic system help make blocked squares, the angel's path, and active areas more understandable?

A — where the angel can move, being highlighted would be essential

the question. Q — Should the tool include helper overlays (e.g highlight reachable squares, show danger zones, visualise path options)?

A — Not essential

1.3.4 Analysis

the question. Q — What kind of analysis or metrics would be most valuable to you after a game simulation? (e.g survival time, number of moves, area explored)

A — Turns moved would be good

the question. Q — Would you find it helpful to have the option to export move sequences or game logs for later analysis?

A — that is not essential but a good suggestion

the question. Q — Would you find it useful to compare different strategies side by side within the tool?

A — No

the question. Q — When playing optimally, would you like the angel to show how it came up with its strategy?

A — In the basic version, the angel should be played by a human player anyway.

1.3.5 Design preferences

the question. Q — What features or tools would make this simulator useful for your research or understanding of the angel problem?

A — I am intending it to be used for demonstration purposes, so a clear interface is important where what is going on is intuitive. I.e. colours

are what you might expect (red for devil?) The most recently placed devil block should be highlighted in some way, and where blocks are placed far apart, there should be some system so the angel and devil players can 'orientate' themselves.

1.4 Modelling

1.4.1 Existing solutions

Oddvar Kloster's demo system is an interactive application built to illustrate his algorithm for a 2-power angel in the angel problem. The system presents a grid display that marks the angel's current position, and highlights all moves the angel has that are disallowed by Kloster's algorithm and includes the option to undo moves. By concentrating exclusively on the 2-power angel, the demo makes it easier to follow the basic movement mechanics and understand how the angel can evade entrapment in a play area. The algorithm evaluates every possible move at each turn and selects those that allow the angel to avoid being trapped, even with a limited move range of two squares. This serves as a single-player game, where the user acts as the devil and tries to trap the angel. The main method of movement throughout the grid is the mouse drag feature. It also includes the option to zoom in and out to increase the speed of movement throughout the infinite grid.

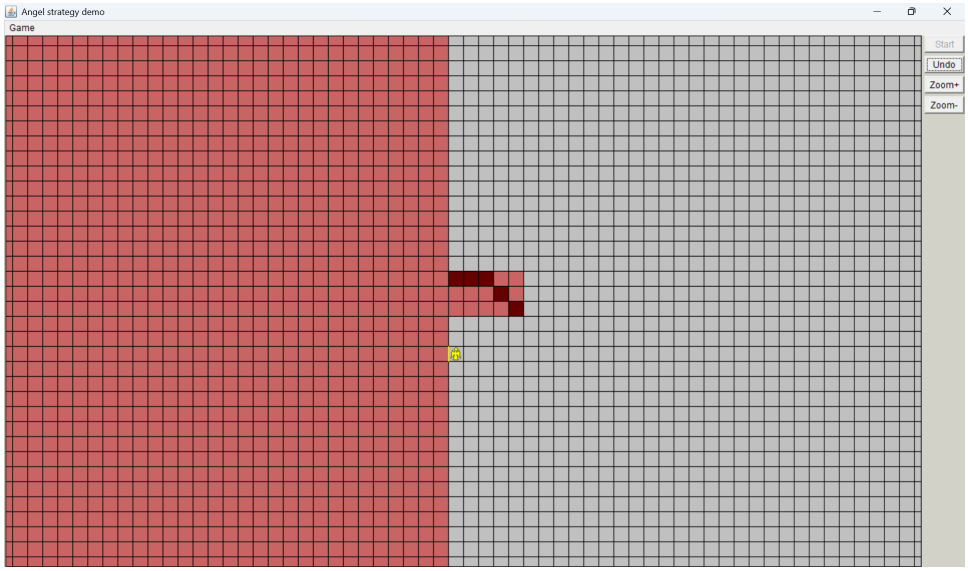
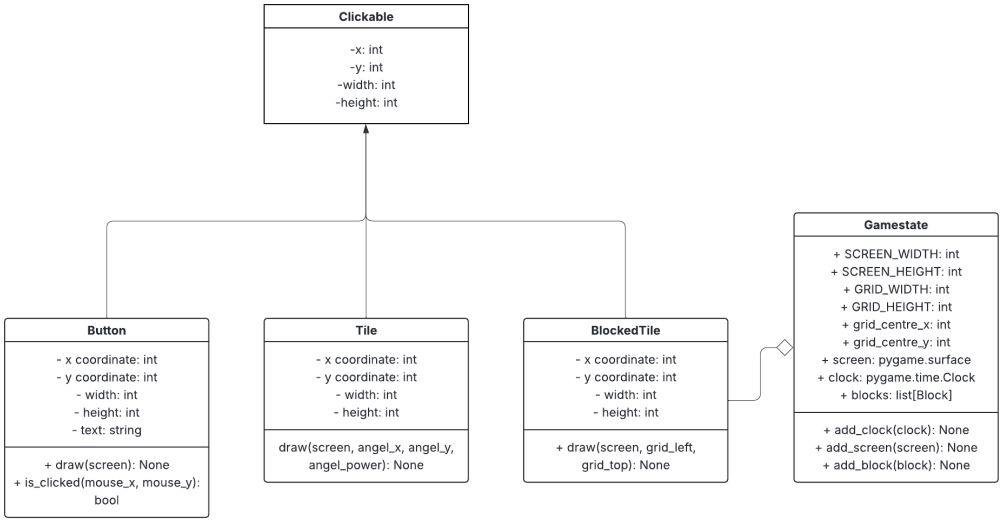


Figure 1.1: An example of the solution made by Kloster

1.4.2 Use of data structures

Below is a UML diagram to display the classes that will be used in the program and the relationships between them.



In Addition, to incorporate the Undo/Redo tasks, a dedicated data class for moves will be used.

1.5 Objectives

1. A menu screen shall appear at program startup, allowing users to configure game settings such as angel power and any movement restrictions
2. The program should operate in a clear turn-based manner for two players, where one is controlling the angel and the other the devil.
3. The program shall display an infinite grid of squares
4. Prior to starting the simulation, users will be able to select the angel's power, which determines the maximum number of squares it can move in one turn.
5. Allow one user to play as an angel, which:
   (a) is to be represented by a distinct icon.
   (b) Users will be able to move by mouse-clicking on a highlighted legal square
6. Allow another player to play as a devil who:

(a) will have moves consisting of placing blocking tiles on the grid by clicking on a target square.

(b) Blocks cannot be placed on the square occupied by the angel.

(c) Will clearly show to either player the location of their last placed block.

7. An undo and redo system will be provided so that users can correct mistakes or attempt different strategies

8. The current coordinates of the angel and the last devil block must be displayed at all times so that users can locate them.

9. The total number of turns taken should be continually updated and shown on the interface.

10. The simulation should end automatically when the angel becomes trapped, with a prompt offering the option to restart or exit the simulation, or the game is mutually ended by both players through an exit button.

1.6 Extention Objectives

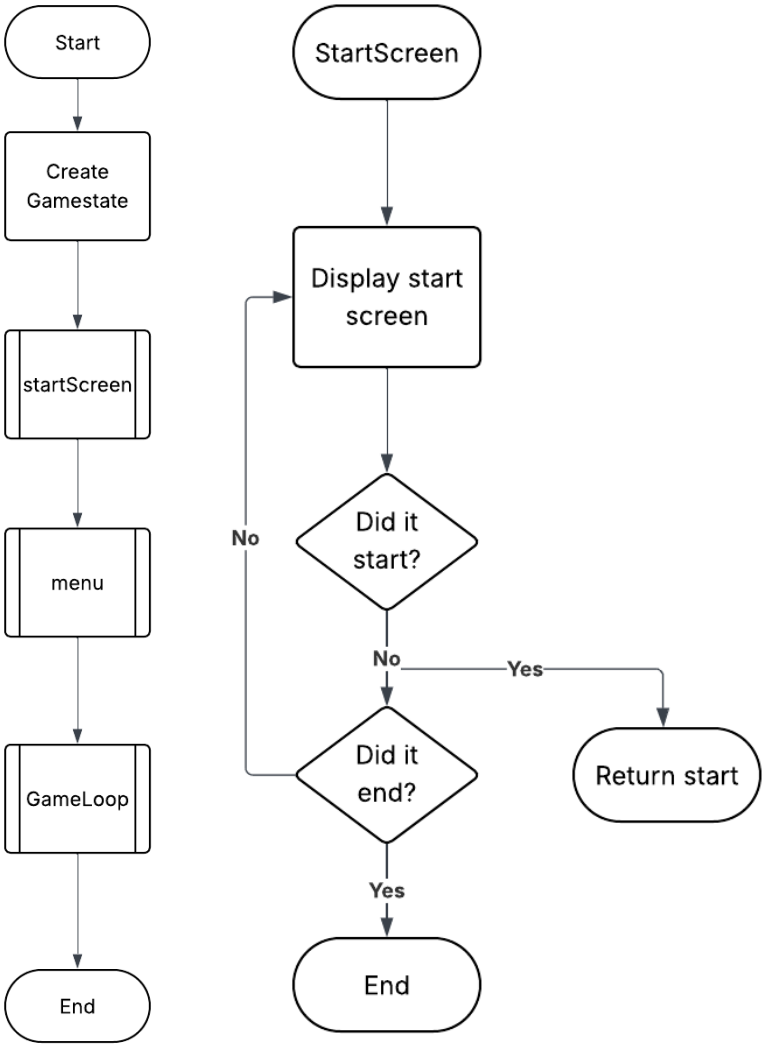
1. The program may offer an optional automated strategy mode for the angel and/or the devil using predefined algorithms

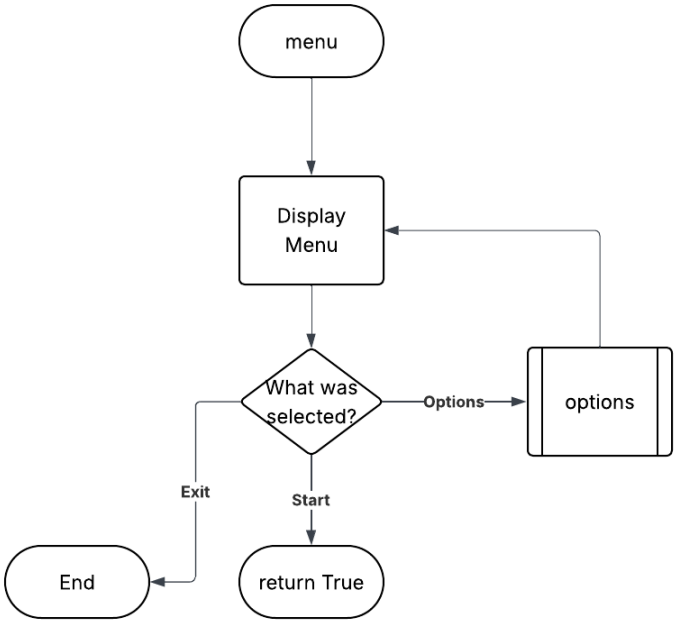
2. The system may provide an option for alternative movement rules (e.g. restricting the angel to only northward moves or requiring moves to be exactly its power).

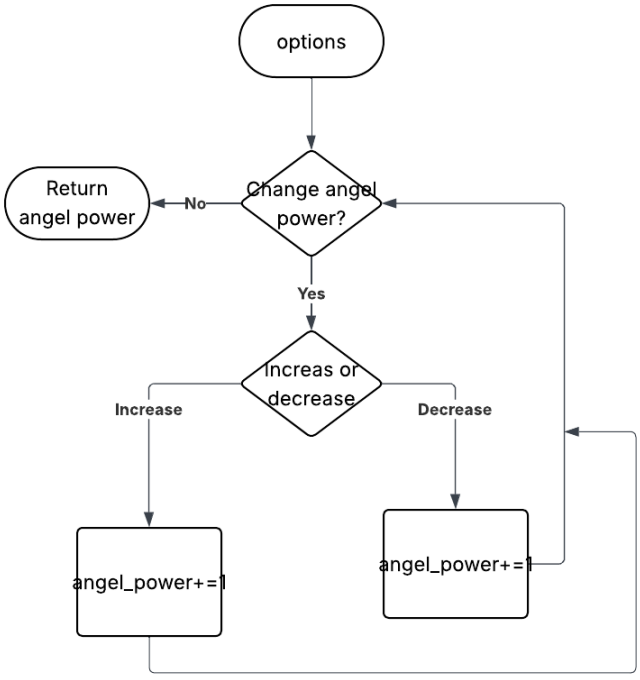
3. An editor may be included to allow users to manually place or remove blocks on the grid, creating custom scenarios during a simulation and save/load them externally.

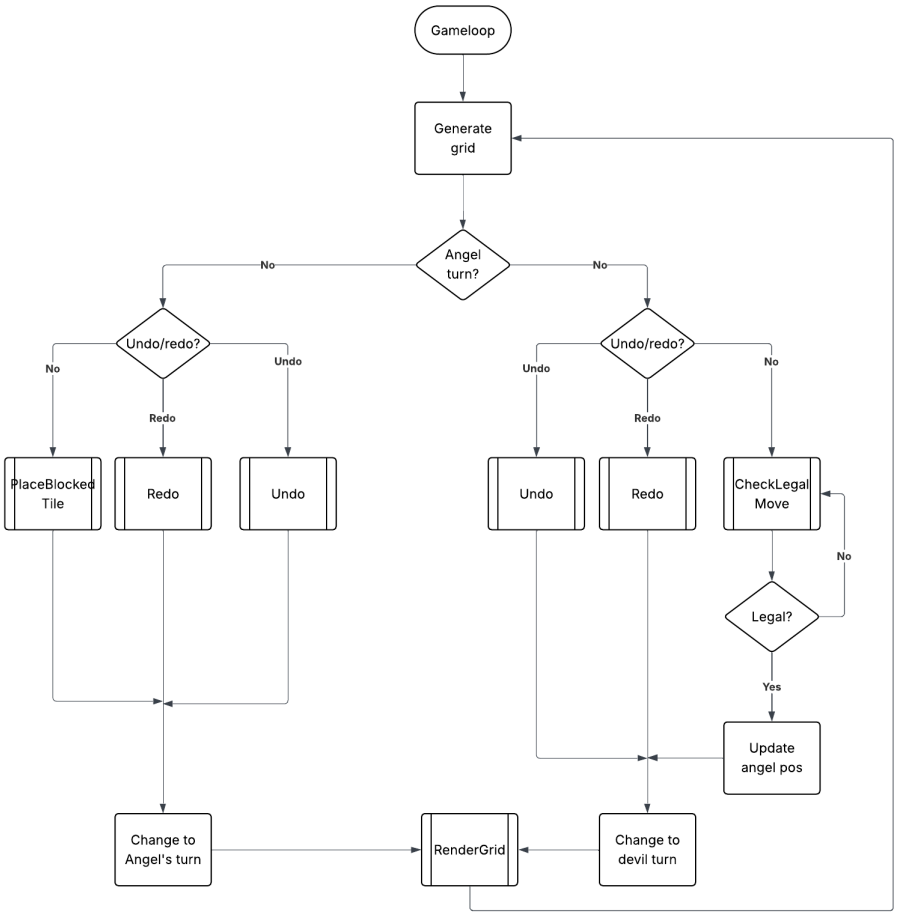
2 Documented Design

2.1 Flow of the program




menu
Display Menu
What was selected?
Options
options
Exit
Start
End
return True


options
Change angel power?
No
Return angel power
Yes
Increas or decrease
Increase
Decrease
angel_power+=1
angel_power+=1



3 Technical Solution

```
import os
import sys

os.environ['PYGAME_HIDE_SUPPORT_PROMPT'] = '1'

import pygame


class Clickable:

    def __init__(self, x, y, width, height):
        self.x = x
        self.y = y
        self.width = width
        self.height = height
```

```
16
17 class Tile(Clickable):
18
19     def __init__(self, x, y, grid_tile_width, grid_tile_height):
20         #x, y are the tiles position on the grid and not the screen
21         super().__init__(x, y, grid_tile_width, grid_tile_height)
22         self.angel_image = pygame.image.load("Images/angel_image.
    png").convert_alpha()
23
24     def draw(self, screen, angel_x, angel_y, angel_power ):
25         ## Check if the tile is within the angel's range
26         ## if the tile is within the angel's range, draw it
27         ## Otherwise, draw it as gray
28         rect = pygame.Rect(self.x * self.width, self.y * self.
    height, self.width, self.height)
29         if (self.x == angel_x and self.y == angel_y):
30             pygame.draw.rect(screen, (255, 251, 0), rect,
    border_radius=5)
31             scaled_image = pygame.transform.scale(self.angel_image,
     (self.width, self.height))
32             screen.blit(scaled_image, rect)
33         elif (self.x >= angel_x - angel_power and self.x <= angel_x
     + angel_power and
34                 self.y >= angel_y - angel_power and self.y <=
    angel_y + angel_power):
35             pygame.draw.rect(screen, (123, 242, 242), rect,
    border_radius=5)
36         else:
37             pygame.draw.rect(screen, (200, 200, 200), rect,
    border_radius=5)
38
39 class BlockedTile(Clickable):
40
41     def __init__(self, x, y, grid_tile_width, grid_tile_height):
42         super().__init__(x, y, grid_tile_width, grid_tile_height)
43
44     def is_on_grid(self, grid_left, grid_top,
    visible_tile_count_width, visible_tile_count_height):
45         # Check if the absolute tile coordinate lies within the
    visible grid
46         return (self.x >= grid_left and
47                 self.x < grid_left + visible_tile_count_width and
48                 self.y >= grid_top and
49                 self.y < grid_top + visible_tile_count_height)
50
51     def draw(self, screen, grid_left, grid_top):
52         # Calculate the tile's position relative to the visible
    grid.
53         visible_x = self.x - grid_left
54         visible_y = self.y - grid_top
55         block_x = visible_x * self.width
56         block_y = visible_y * self.height
57         pygame.draw.rect(screen, (125, 19, 19), (block_x, block_y,
    self.width, self.height), border_radius=5)
58 class Button(Clickable):
59
60     def __init__(self, x, y, width, height, text):
```

```
61          super().__init__(x, y, width, height)
62          self.text = text
63
64      def draw(self, screen):
65          pygame.draw.rect(screen, (0, 230, 255), (self.x-self.width
    // 2, self.y-self.height // 2, self.width, self.height),
    border_radius=5)
66          font = pygame.font.Font(None, 36)
67          text_surface = font.render(self.text, True, (0, 0, 0))
68          text_rect = text_surface.get_rect(center=(self.x, self.y))
69          screen.blit(text_surface, text_rect)
70
71      def is_clicked(self, mouse_x, mouse_y):
72          return (self.x - self.width // 2 <= mouse_x <= self.x +
    self.width // 2 and
73                  self.y - self.height // 2 <= mouse_y <= self.y +
    self.height // 2)
74
75  class Move:
76      def __init__(self, move_type, tile_x, tile_y):
77          """
78          move_type: "angel" or "block"
79          tile_x, tile_y: absolute tile coordinates at which the move
     occurred.
80          """
81          self.move_type = move_type
82          self.tile_x = tile_x
83          self.tile_y = tile_y
84
85  class GameState:
86      def __init__(self):
87          # Screen dimensions
88          self.SCREEN_WIDTH = 800
89          self.SCREEN_HEIGHT = 600
90
91          # Grid area dimensions (e.g. 600x600 for the grid, leaving
    200 for UI)
92          self.GRID_AREA_WIDTH = 600
93          self.GRID_AREA_HEIGHT = 600
94
95          # Grid settings (10x10 grid)
96          self.GRID_COLS = 10
97          self.GRID_ROWS = 10
98          self.tile_width = self.GRID_AREA_WIDTH // self.GRID_COLS
99          self.tile_height = self.GRID_AREA_HEIGHT // self.GRID_ROWS
100
101         # Offsets: these are absolute tile indices for the top-left
     tile of the grid view.
102         self.grid_left = 0
103         self.grid_top = 0
104
105         # Angel's absolute position (tile coordinates) and power
106         self.angel_power = 1
107         self.angel_x = 0
108         self.angel_y = 0
109
110         # List of BlockedTiles (using absolute coordinates)
```

```
111         self.blocks = []
112 
113         # Move stacks for undo/redo; each element is a Move object.
114         self.undo_stack = []
115         self.redo_stack = []
116 
117         # UI buttons for undo and redo (positioned to the right of
    the grid area)
118         self.undo_button = Button(700, 160, 80, 40, "Undo")
119         self.redo_button = Button(700, 220, 80, 40, "Redo")
120     def add_clock(self, clock):
121         self.clock = clock
122 
123     def add_screen(self, screen):
124         self.screen = screen
125 
126     def add_block(self, block):
127         self.blocks.append(block)
128 
129 def main():
130     game_state = GameState()
131 
132     pygame.init()
133     pygame.font.init()
134     pygame.display.set_caption('Angel␣Problem')
135     icon = pygame.image.load('Images/angel_icon.png')
136     pygame.display.set_icon(icon)
137     screen = pygame.display.set_mode((game_state.SCREEN_WIDTH,
    game_state.SCREEN_HEIGHT))
138     clock = pygame.time.Clock()
139 
140     game_state.add_screen(screen)
141     game_state.add_clock(clock)
142 
143     end = startScreen(game_state)
144     start = True
145     while not end and start:
146         start, end = menu(game_state)
147         print("ended␣menu")
148         if start:
149             game_state, start, end = gameloop(game_state)
150     exitGame()
151 
152 def startScreen(game_state):
153     # Fill half of the screen with white and the other half with
    red
154     SCREEN_WIDTH = game_state.SCREEN_WIDTH
155     SCREEN_HEIGHT = game_state.SCREEN_HEIGHT
156     clock = game_state.clock
157     screen = game_state.screen
158     screen.fill((255, 255, 255))
159     half_screen = SCREEN_WIDTH // 2
160     pygame.draw.rect(screen, (255, 0, 0), pygame.Rect(0,0,
    half_screen, SCREEN_HEIGHT), border_radius=5)
161     title = pygame.font.Font(None, 74).render("Angel␣Problem", True
    , (0, 0, 0))
162     title_rect = title.get_rect(center=(half_screen, SCREEN_HEIGHT
```

```
    // 2 - 100))
    screen.blit(title, title_rect)
    startButton = Button(half_screen, SCREEN_HEIGHT // 2 - 25, 100, 50, "Start")
    credits_text = pygame.font.Font(None, 12).render("Credits to Iconka & sodiqmahmud46 for all images", True, (0, 0, 0))
    credits_text_rect = credits_text.get_rect(center=(SCREEN_WIDTH -100, SCREEN_HEIGHT-12))
    screen.blit(credits_text,credits_text_rect)
    start = False
    end = False
    while not (start or end):
        for event in pygame.event.get():
            if event.type == pygame.QUIT:
                end = True
            if event.type == pygame.MOUSEBUTTONDOWN:
                mouse_x, mouse_y = event.pos
                if startButton.is_clicked(mouse_x, mouse_y):
                    start = True

        screen.fill((255, 255, 255))
        pygame.draw.rect(screen, (255, 0, 0), pygame.Rect(0,0, half_screen, SCREEN_HEIGHT), border_radius=5)
        startButton.draw(screen)
        screen.blit(title, title_rect)
        screen.blit(credits_text,credits_text_rect)
        pygame.display.update()
        clock.tick(60)
    if end:
        return True
    else:
        return False

def menu(game_state):
    SCREEN_WIDTH = game_state.SCREEN_WIDTH
    SCREEN_HEIGHT = game_state.SCREEN_HEIGHT
    clock = game_state.clock
    screen = game_state.screen
    clearScreen(screen, (230, 230, 230))
    play_button = Button(SCREEN_WIDTH // 2, SCREEN_HEIGHT // 2 - 25, 100, 50, "Play")
    options_button = Button(SCREEN_WIDTH // 2, SCREEN_HEIGHT // 2 + 25, 100, 50, "Options")
    exit_button = Button(SCREEN_WIDTH // 2, SCREEN_HEIGHT // 2 + 75, 100, 50, "Exit")
    end = False
    start = False
    while not(end or start):
        for event in pygame.event.get():
            if event.type == pygame.QUIT:
                end = True
            if event.type == pygame.MOUSEBUTTONDOWN:
                mouse_x, mouse_y = event.pos
                if (play_button.is_clicked(mouse_x, mouse_y)):
                    start = True
                elif (options_button.is_clicked(mouse_x, mouse_y)):
                    game_state = options(game_state)
```

```
                elif (exit_button.is_clicked(mouse_x, mouse_y)):
                    end = True
        clearScreen(screen)
        play_button.draw(screen)
        options_button.draw(screen)
        exit_button.draw(screen)
        pygame.display.update()
        clock.tick(60)
    return start, end

def options(game_state):
    # Add the option to change the angel's power using up and down buttons
    SCREEN_WIDTH = game_state.SCREEN_WIDTH
    SCREEN_HEIGHT = game_state.SCREEN_HEIGHT
    clock = game_state.clock
    screen = game_state.screen
    angel_power = game_state.angel_power
    clearScreen(screen, (230, 230, 230))

    title = pygame.font.Font(None, 74).render("Options", True, (0, 0, 0))
    title_rect = title.get_rect(center=(SCREEN_WIDTH // 2, SCREEN_HEIGHT // 2 - 50))
    screen.blit(title, title_rect)
    angel_power_text = pygame.font.Font(None, 36).render(f"Angel Power: {angel_power}", True, (0, 0, 0))
    angel_power_text_rect = angel_power_text.get_rect(center=(SCREEN_WIDTH // 2, SCREEN_HEIGHT // 2))
    screen.blit(angel_power_text, angel_power_text_rect)
    up_button = Button(SCREEN_WIDTH // 2, SCREEN_HEIGHT // 2 + 50, 100, 50, "Up")
    down_button = Button(SCREEN_WIDTH // 2, SCREEN_HEIGHT // 2 + 100, 100, 50, "Down")
    back_button = Button(SCREEN_WIDTH // 2, SCREEN_HEIGHT // 2 + 150, 100, 50, "Back")
    loop = True
    while loop:
        clearScreen(screen, (230, 230, 230))
        back_button.draw(screen)
        up_button.draw(screen)
        down_button.draw(screen)
        angel_power_text = pygame.font.Font(None, 36).render(f"Angel Power: {angel_power}", True, (0, 0, 0))
        angel_power_text_rect = angel_power_text.get_rect(center=(SCREEN_WIDTH // 2, SCREEN_HEIGHT // 2))
        screen.blit(angel_power_text, angel_power_text_rect)

        for event in pygame.event.get():
            if event.type == pygame.QUIT:
                loop = False
            if event.type == pygame.MOUSEBUTTONDOWN:
                mouse_x, mouse_y = event.pos
                if (up_button.is_clicked(mouse_x, mouse_y)):
                    angel_power += 1
                    game_state.angel_power = angel_power
                elif (down_button.is_clicked(mouse_x, mouse_y)):
```

```
                    if (angel_power > 1):
                        angel_power -= 1
                        game_state.angel_power = angel_power
                    else:
                        angel_power = 1
                elif (back_button.is_clicked(mouse_x, mouse_y)):
                    loop = False

        pygame.display.update()
        clock.tick(60)
    return game_state


# A helper to center the grid view on a given move.
def center_grid_on_move(game_state, move_x, move_y):
    half_cols = game_state.GRID_COLS // 2
    half_rows = game_state.GRID_ROWS // 2
    game_state.grid_left = move_x - half_cols
    game_state.grid_top = move_y - half_rows



# In your gameloop, you can directly record moves and update state.
def gameloop(game_state):
    SCREEN_WIDTH = game_state.SCREEN_WIDTH
    SCREEN_HEIGHT = game_state.SCREEN_HEIGHT
    clock = game_state.clock
    screen = game_state.screen
    grid_width = game_state.tile_width  # using tile_width now derived from the grid area.
    grid_height = game_state.tile_height
    current_player = "Angel"
    turn_number = 1
    grid = [[Tile(i, j, grid_width, grid_height) for j in range(game_state.GRID_ROWS)]
            for i in range(game_state.GRID_COLS)]  # 10x10 grid as before

    menu_button = Button(700, 650, 80, 40, "Menu")
    exit_button = Button(700, 700, 80, 40, "Quit")
    angel_button = Button(700, 500, 140, 40, "Goto␣Angel")
    block_button = Button(700, 550, 140, 40, "Goto␣Block")

    win = False
    while not win:
        for event in pygame.event.get():
            if event.type == pygame.QUIT:
                return game_state, False, True

            if event.type == pygame.MOUSEBUTTONDOWN:
                mouse_x, mouse_y = event.pos
                # Check if click is in the UI panel
                if mouse_x > game_state.GRID_AREA_WIDTH:
                    # UI area for undo/redo
                    if game_state.undo_button.is_clicked(mouse_x, mouse_y):
                        game_state = undoMove(game_state)
```

```
312                         if game_state.undo_stack:
313                             current_player = "Angel" if turn_number
    % 2 == 0 else "Devil"
314                             turn_number -= 1
315                     elif game_state.redo_button.is_clicked(mouse_x,
    mouse_y):
316                         game_state = redoMove(game_state)
317                         if game_state.redo_stack:
318                             current_player = "Angel" if turn_number
    % 2 == 0 else "Devil"
319                             turn_number += 1
320                     elif menu_button.is_clicked(mouse_x,mouse_y):
321                         game_state = GameState()
322                         game_state.add_clock(clock)
323                         game_state.add_screen(screen)
324                         return game_state, True, False
325                     elif exit_button.is_clicked(mouse_x,mouse_y):
326                         return game_state, False, True
327 
328                     elif angel_button.is_clicked(mouse_x,mouse_y):
329                         game_state.grid_left = game_state.angel_x -
    game_state.GRID_ROWS//2
330                         game_state.grid_top = game_state.angel_y -
    game_state.GRID_COLS//2
331                     elif block_button.is_clicked(mouse_x,mouse_y):
332                         game_state.grid_left = game_state.blocks
    [-1].x - game_state.GRID_ROWS//2 if  game_state.blocks[-1] else
    game_state.grid_left
333                         game_state.grid_top = game_state.blocks
    [-1].y - game_state.GRID_COLS//2 if  game_state.blocks[-1] else
    game_state.grid_top
334 
335 
336                 else:
337                     # Process grid clicks (angel move or block
    placement).
338                     if current_player == "Angel":
339                         # Convert click (in pixels) into an
    absolute tile coordinate.
340                         tile_x = (mouse_x // grid_width) +
    game_state.grid_left
341                         tile_y = (mouse_y // grid_height) +
    game_state.grid_top
342                         print("clicked␣(abs):", tile_x, tile_y)
343                         print("angel␣at:", game_state.angel_x,
    game_state.angel_y)
344                         # use checkLegalMove with appropriate
    coordinate conversion.
345                         if checkLegalMove(game_state, tile_x -
    game_state.grid_left, tile_y - game_state.grid_top):
346                             # Record the angel move.
347                             move = Move("angel", tile_x, tile_y)
348                             game_state.undo_stack.append(move)
349                             game_state.redo_stack.clear()
350 
351                             # Update angel's state.
352                             game_state.angel_x = tile_x
```

```
                            game_state.angel_y = tile_y
                            current_player = "Devil"
                            turn_number += 1
                    elif current_player == "Devil":
                        # For block placement, convert click coordinates:
                        tile_x = (mouse_x // grid_width) + game_state.grid_left
                        tile_y = (mouse_y // grid_height) + game_state.grid_top

                        # Check that tile is available and within the grid.
                        if (tile_x >= game_state.grid_left and tile_x < game_state.grid_left + game_state.GRID_COLS and
                            tile_y >= game_state.grid_top and tile_y < game_state.grid_top + game_state.GRID_ROWS):
                            # Prevent duplicate block placements and block if angel is here.
                            if not any(b.x == tile_x and b.y == tile_y for b in game_state.blocks) and \
                                not (tile_x == game_state.angel_x and tile_y == game_state.angel_y):
                                new_block = BlockedTile(tile_x, tile_y, grid_width, grid_height)
                                move = Move("block", tile_x, tile_y)
                                game_state.undo_stack.append(move)
                                game_state.redo_stack.clear()

                                game_state.blocks.append(new_block)
                                current_player = "Angel"
                                turn_number += 1
                                if checkWin(game_state):
                                    win = True

            if event.type == pygame.KEYDOWN:
                if event.key in [pygame.K_w, pygame.K_a, pygame.K_s, pygame.K_d, pygame.K_LEFT, pygame.K_RIGHT, pygame.K_UP, pygame.K_DOWN]:
                    game_state = moveGrid(game_state, event.key)

        renderGrid(screen, game_state, grid, current_player, turn_number, menu_button, exit_button, angel_button, block_button)
    #display the win screen and return to the menu
    while True:
        clearScreen(screen, (200,0,0))
        font = pygame.font.Font(None, 74)
        win_text = font.render("Devil Wins!", True, (255, 255, 255))
        win_rect = win_text.get_rect(center=(SCREEN_WIDTH // 2, SCREEN_HEIGHT // 2-100))
        screen.blit(win_text,win_rect)
        # Add a button to go back to the menu
        back_button = Button(SCREEN_WIDTH // 2, SCREEN_HEIGHT // 2 + 50, 100, 50, "Menu")
```

```
        back_button.draw(screen)
        for event in pygame.event.get():
            if event.type == pygame.QUIT:
                return game_state, False, True
            elif event.type == pygame.MOUSEBUTTONDOWN:
                mouse_x, mouse_y = event.pos
                if back_button.is_clicked(mouse_x, mouse_y):
                    # Reset game state for the next round
                    game_state = GameState()
                    game_state.add_screen(screen)
                    game_state.add_clock(clock)
                    return game_state, True, False

        pygame.display.update()
        clock.tick(60)




# The undoMove function is where we now center the grid.
def undoMove(game_state):
    if not game_state.undo_stack:
        print("Nothing to undo.")
        return game_state

    move = game_state.undo_stack.pop()

    if move.move_type == "angel":
        # Look back at the last angel move if there is one.
        last_angel_move = None
        for m in reversed(game_state.undo_stack):
            if m.move_type == "angel":
                last_angel_move = m
                break
        if last_angel_move:
            game_state.angel_x = last_angel_move.tile_x
            game_state.angel_y = last_angel_move.tile_y
            center_grid_on_move(game_state, last_angel_move.tile_x,
     last_angel_move.tile_y)
        else:
            # Default state if no previous angel move.
            game_state.angel_x = 0
            game_state.angel_y = 0
            center_grid_on_move(game_state, 0, 0)
    elif move.move_type == "block":
        # Remove the block that matches this move.
        game_state.blocks = [b for b in game_state.blocks if not (b
    .x == move.tile_x and b.y == move.tile_y)]
        # Center on the most recent move in the undo stack.
        if game_state.undo_stack:
            last = game_state.undo_stack[-1]
            center_grid_on_move(game_state, last.tile_x, last.
    tile_y)

    game_state.redo_stack.append(move)
    return game_state

```

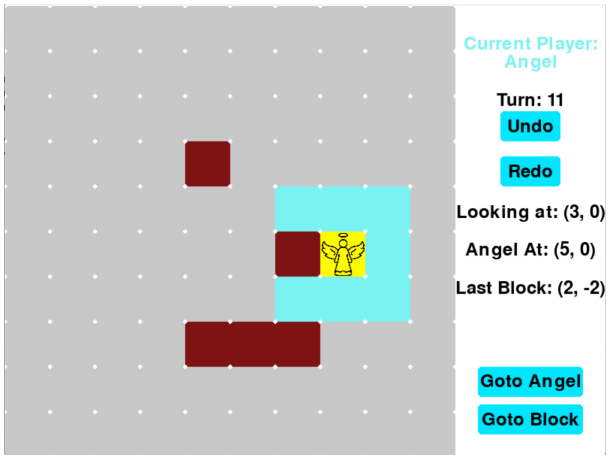
```
446 def redoMove(game_state):
447     if not game_state.redo_stack:
448         print("Nothing to redo.")
449         return game_state
450 
451     move = game_state.redo_stack.pop()
452 
453     if move.move_type == "angel":
454         game_state.angel_x = move.tile_x
455         game_state.angel_y = move.tile_y
456         center_grid_on_move(game_state, move.tile_x, move.tile_y)
457     elif move.move_type == "block":
458         new_block = BlockedTile(move.tile_x, move.tile_y,
    game_state.tile_width, game_state.tile_height)
459         game_state.blocks.append(new_block)
460         center_grid_on_move(game_state, move.tile_x, move.tile_y)
461 
462     game_state.undo_stack.append(move)
463     return game_state
464 
465 
466 def renderGrid(screen, game_state, grid, current_player,
    turn_number, menu_button, exit_button, angel_button,
    block_button):
467     clock = game_state.clock
468     screen.fill((255, 255, 255))
469     # Calculate visible angel coordinates based on the current grid
     offset:
470     visible_angel_x = game_state.angel_x - game_state.grid_left
471     visible_angel_y = game_state.angel_y - game_state.grid_top
472 
473     for i in range(10):
474         for j in range(10):
475             grid[i][j].draw(screen, visible_angel_x,
    visible_angel_y, game_state.angel_power)
476 
477     for block in game_state.blocks:
478         if block.is_on_grid(game_state.grid_left, game_state.
    grid_top, 10, 10):
479             block.draw(screen, game_state.grid_left, game_state.
    grid_top)
480 
481 
482     # Show the turn of the current player in the right panel
483     font = pygame.font.Font(None, 36)
484 
485     if current_player == "Angel":
486         player_text = font.render("Current Player:", True, (123,
    242, 242))
487     else:
488         player_text = font.render("Current Player:", True, (255, 0,
     0))
489     player_rect = player_text.get_rect(center=(game_state.
    GRID_AREA_WIDTH + 100, 50))
490     screen.blit(player_text, player_rect)
491 
492     if current_player == "Angel":
```

```
493             player_text = font.render("Angel", True, (123, 242, 242))
494         else:
495             player_text = font.render("Devil", True, (255, 0, 0))
496         player_rect = player_text.get_rect(center=(game_state.
            GRID_AREA_WIDTH + 100, 75))
497         screen.blit(player_text, player_rect)
498 
499         turn_text = font.render(f"Turn: {turn_number}", True, (0, 0, 0)
            )
500         turn_rect = turn_text.get_rect(center=(game_state.
            GRID_AREA_WIDTH + 100, 125))
501         screen.blit(turn_text, turn_rect)
502 
503         coordinates_text = font.render(f"Looking at: ({game_state.
            grid_left + game_state.GRID_COLS // 2}, {game_state.grid_top + 
            game_state.GRID_ROWS // 2})", True, (0, 0, 0))
504         coordinates_rect = coordinates_text.get_rect(center=(game_state
            .GRID_AREA_WIDTH + 100, 275))
505         screen.blit(coordinates_text, coordinates_rect)
506 
507         angel_coordinates_text = font.render(f"Angel At: ({game_state.
            angel_x}, {game_state.angel_y})", True, (0, 0, 0))
508         angel_coordinates_rect = angel_coordinates_text.get_rect(center
            =(game_state.GRID_AREA_WIDTH + 100, 325))
509         screen.blit(angel_coordinates_text, angel_coordinates_rect)
510 
511         last_block_coordinates_text = font.render(f"Last Block: {(
            game_state.blocks[-1].x,game_state.blocks[-1].y) if game_state.
            blocks else None}", True, (0, 0, 0))
512         last_block_coordinates_rect = last_block_coordinates_text.
            get_rect(center=(game_state.GRID_AREA_WIDTH + 100, 375))
513         screen.blit(last_block_coordinates_text,
            last_block_coordinates_rect)
514 
515 
516 
517         # Draw the undo and redo buttons
518         game_state.undo_button.draw(screen)
519         game_state.redo_button.draw(screen)
520 
521         menu_button.draw(screen)
522         exit_button.draw(screen)
523         angel_button.draw(screen)
524         block_button.draw(screen)
525 
526         pygame.display.update()
527         clock.tick(60)
528 
529 def moveGrid(game_state, key):
530     match key:
531         case pygame.K_w | pygame.K_UP:
532             game_state.grid_top -= 1
533         case pygame.K_a | pygame.K_LEFT:
534             game_state.grid_left -= 1
535         case pygame.K_s | pygame.K_DOWN:
536             game_state.grid_top += 1
537         case pygame.K_d | pygame.K_RIGHT:
```

```
            game_state.grid_left += 1
    return game_state

def placeBlockedTile(game_state, mouse_x, mouse_y, blocked_tiles):
    grid_width = game_state.GRID_WIDTH
    grid_height = game_state.GRID_HEIGHT
    grid_offset_x = game_state.grid_left  # these are tile indices
    grid_offset_y = game_state.grid_top

    # Convert pixel click to an absolute tile coordinate
    tile_x = (mouse_x // grid_width) + grid_offset_x
    tile_y = (mouse_y // grid_height) + grid_offset_y

    # Check that the clicked tile falls within the visible grid (10
    x10)
    if (tile_x >= grid_offset_x and tile_x < grid_offset_x + 10 and
        tile_y >= grid_offset_y and tile_y < grid_offset_y + 10):

        # Check that no blocked tile already exists at this
    absolute coordinate
        if not any(block.x == tile_x and block.y == tile_y for
    block in blocked_tiles):
            # Also, prevent placing a block on the angel's position
     (absolute coordinates)
            if not (tile_x == game_state.angel_x and tile_y ==
    game_state.angel_y):
                new_block = BlockedTile(tile_x, tile_y, grid_width,
     grid_height)
                blocked_tiles.append(new_block)
                game_state.add_block(new_block)
    return game_state, blocked_tiles

def checkLegalMove(game_state, tile_x, tile_y):
    # Convert the clicked relative coordinates into an absolute
    coordinate
    clicked_abs_x = tile_x + game_state.grid_left
    clicked_abs_y = tile_y + game_state.grid_top

    # Use the absolute angel position as stored in game_state
    angel_abs_x = game_state.angel_x
    angel_abs_y = game_state.angel_y

    # Assuming blocked_tiles are stored in absolute coordinates; if
     not, adjust similarly.
    blocked_tiles = game_state.blocks

    if (tile_x >= 0 and tile_x < 10 and tile_y >= 0 and tile_y < 10
     and
        not any(block.x == clicked_abs_x and block.y ==
    clicked_abs_y for block in blocked_tiles) and
        not (clicked_abs_x == angel_abs_x and clicked_abs_y ==
    angel_abs_y) and
        clicked_abs_x >= angel_abs_x - game_state.angel_power and
        clicked_abs_x <= angel_abs_x + game_state.angel_power and
        clicked_abs_y >= angel_abs_y - game_state.angel_power and
        clicked_abs_y <= angel_abs_y + game_state.angel_power):
        print("legal␣move")
```

```
584         return True
585
586     return False
587 def checkWin(game_state):
588     ax = game_state.angel_x  # angel's absolute x coordinate
589     ay = game_state.angel_y  # angel's absolute y coordinate
590     p = game_state.angel_power
591
592     # Loop over every candidate tile in the angel's move range.
593     # For each candidate (dx, dy) offset from the angel,
594     # skip the angel's current position.
595     for dx in range(-p, p + 1):
596         for dy in range(-p, p + 1):
597             if dx == 0 and dy == 0:
598                 continue  # skip the current position
599             candidate_x = ax + dx
600             candidate_y = ay + dy
601             # Check the candidate is not blocked:
602             blocked = any(block.x == candidate_x and block.y ==
    candidate_y
603                           for block in game_state.blocks)
604             if not blocked:
605                 # Found at least one legal move
606                 return False
607
608     # If every candidate move is blocked, the angel is trapped.
609     print("Angel␣is␣trapped!␣Devil␣wins!")
610     return True
611
612 def clearScreen(screen,colour=(255, 255, 255)):
613     screen.fill(colour)
614
615 def exitGame():
616     pygame.quit()
617     sys.exit()
618
619 if __name__=="__main__":
620     main()
```

4 Testing



5 Evaluation

So far I have fully implemented the internal workings for the Angel Problem and have met all of the objectives given.

6 references