ĐẠI HỌC QUỐC GIA THÀNH PHỐ HỒ CHÍ MINH

TRƯỜNG ĐẠI HỌC KHOA HỌC TỰ NHIÊN

Đồ án
Xây dựng trò chơi Mummy Maze

Báo cáo kĩ thuật
CSC10013 - Cơ sở lập trình cho Trí tuệ nhân tạo

Sinh viên thực hiện:

Trần Đình Hưng

Vương Thành Phát

Lê Thanh Phi

Nguyễn Quốc Thái

Giảng viên hướng dẫn:

Lê Thanh Tùng

Trần Hoàng Quân

Thành phố Hồ Chí Minh, Ngày 11 tháng 1 năm 2026

MỤC LỤC

MỤC LỤC **1**

1 THÔNG TIN NHÓM **4**

2 Thuật toán và cách triển khai trong game **5**

2.1 Giới thiệu chung 5

2.2 Tổ chức mã nguồn 5

2.2.1 Cấu trúc thư mục (theo file tree hiện tại) 5

2.2.2 Vai trò từng nhóm file (bám sát naming trong code) 6

2.2.3 Điều phối lượt chơi trong `main.py` 7

2.3 Các class chính và cách tổ chức 9

2.3.1 `Cell`: mô hình ô lưới và tường/cổng 9

2.3.2 `Player`: input, animation, smooth slide 9

2.3.3 `Enemy`: AI theo mode, đi 1 hoặc 2 bước, có `uid` 9

2.3.4 `Gamestate`: state chung, items, undo history, solution, gate animation . . 10

2.4 Undo và Restart 11

2.4.1 Tư tưởng triển khai 11

2.4.2 `make_snapshot()`: snapshot gồm những gì 11

2.4.3 `apply_snapshot()`: restore state và reset các cờ movement 12

2.4.4 Undo trong runtime (liên hệ với `main.py`) 12

2.4.5 Restart 13

2.5 Sinh màn chơi (Generate game) 13

2.5.1 Pipeline tổng trong `generate_game()` 13

2.5.2 Prim wall generate: `generate_walls_pr()` 15
2.6 BFS path finding và kiểm tra solvable 17
2.6.1 BFS trong game không chỉ là BFS theo ô 17
2.6.2 State mà BFS đang xét (đúng theo code) 17
2.6.3 Transition trong BFS (mỗi bước tương ứng 1 lượt player) 17
2.6.4 Giả lập enemy turn ngay trong BFS 18
2.6.5 Reconstruct solution 20
2.7 Hai thuật toán đường đi chính của quái 20
2.7.1 Manhattan greedy: `new_enemy_position()` 20
2.7.2 BFS (impossible mode): `impossible_mode_move()` 20
2.7.3 Kết nối với cơ chế 2 bước/turn 20
2.8 Tổ chức level JSON và save JSON 21
2.8.1 Level JSON (assets/map/*) 21
2.8.2 Save JSON (local profile, `saveload.py`) 22
2.9 Sprite sheet, animation, và hệ thống skin 22
2.9.1 Quy ước sprite hướng và frame 22
2.9.2 Cắt sprite sheet kiểu grid (legacy): `add_sprite_frames()` 23
2.9.3 Chạy animation + slide 23

3 UI INTRODUCTION **24**
3.1 Sơ lược giao diện game 24
3.1.1 Tổng quan trò chơi 24
3.1.2 Công nghệ sử dụng 24
3.2 Các chức năng của game: 25
3.2.1 Đồ họa và hoạt ảnh 25
3.2.2 Hệ thống Quản lý tài khoản: 26
3.2.3 Giao diện Menu chính 27

3.2.4 Lựa chọn chế độ chơi 29
3.2.5 Giao diện Màn hình chơi 30
3.2.6 Giao diện khi thất bại 32

CHƯƠNG 1

THÔNG TIN NHÓM

1. **Lê Thanh Phi** (hoàn thành 100%)
   - Xây dựng thuật toán.
   - Trình bày quá trình phát triển game.
2. **Trần Đình Hưng** (hoàn thành 100%)
   - Dựng và edit video
   - Trích tài liệu tham khảo.
3. **Nguyễn Quốc Thái** (hoàn thành 100%)
   - Xây dựng thuật toán.
   - Phát triển thuật toán.
   - Trình bày các thuật toán đã tìm hiểu & triển khai trong game.
4. **Vương Thành Phát** (hoàn thành 100%)
   - Thiết kế giao diện cho ứng dụng.
   - Giới thiệu giao diện người dùng.

CHƯƠNG 2

Thuật toán và cách triển khai trong game

2.1. Giới thiệu chung

Mummy Maze được xây dựng bằng Python và Pygame theo kiểu **grid-based turn game**: player đi **1 hành động**, sau đó enemy đi lần lượt theo lượt. Ngoài Pygame, game dùng **JSON** để lưu **level** (assets/map/) và **save file** (local profile), nhằm giúp nhóm mở rộng nội dung mà không phải sửa gameplay core.

2.2. Tổ chức mã nguồn

2.2.1. Cấu trúc thư mục (theo file tree hiện tại)

```
game/
  main.py
  saveload.py
  sprite_cutting.py
  variable.py
  assets/
    characters/  font/  images/  map/  music/  screen/  sounds/  sprite_sheet/
entity/
  entity.py
module/
  gamestate.py
  module.py
screen/
  button.py
```

```
  login.py
skin_module/
  skin_loader.py
  skin_screen.py
utils/
```

2.2.2. Vai trò từng nhóm file (bám sát naming trong code)

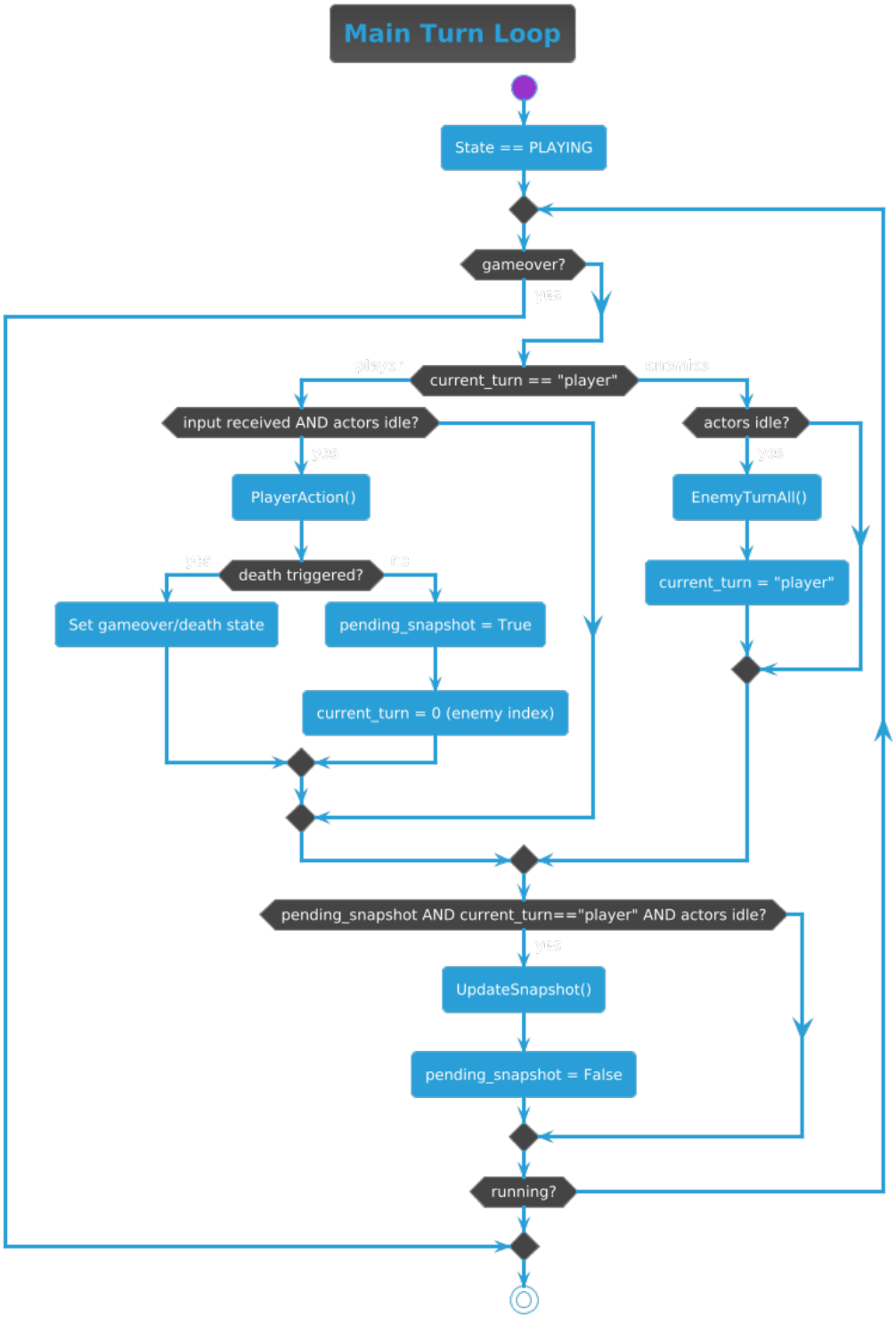
- `game/main.py`: **state machine** tổng (LOGIN / SELECTION / PLAYING / OPTIONS / CHARACTER_SELECT / LOSE_MENU...), vòng lặp chính, bắt input, gọi `player.move()`, chạy **turn system**, gọi **save/load**, điều phối UI và gameplay.
- `game/variable.py`: cấu hình runtime (CELL_SIZE, FPS, ROWS/COLS theo grid, scale, màu...).
- `entity/entity.py`: định nghĩa **Cell, Player, Enemy**. Đây là nơi có logic di chuyển từng entity và hệ animation/slide (render_dx/render_dy, move_timer/move_total, is_moving).
- `module/module.py`: **thuật toán lõi**:
  - snapshot: `make_snapshot()`, `apply_snapshot()`
  - generate: `generate_game()`, `generate_walls_pr()` (Prim)
  - BFS solvable + solution: `path_finding()`, `is_playable()`
  - AI quái: `new_enemy_position()` (Manhattan), `impossible_mode_move()` (BFS)
  - đọc map JSON: `read_map_json()`
  - xử lý tile đặc biệt: `check_special_tiles()`, `toggle_gates()`
- `module/gamestate.py`: class `Gamestate` giữ **toàn bộ state gameplay dùng chung** (mode, level/chapter, keys/traps/gates_h, storedmove/initpos, solution, gate_anim_state, cờ gameover, trạng thái death animation...).
- `game/saveload.py`: build cấu trúc save JSON, serialize game state (player/enemies/items/-gates/move history), lưu local.

- `screen/*`: các màn UI (button/menu/login). Mục tiêu: gọi hàm gameplay chứ không nhét gameplay vào UI.
- `skin_module/*`: hệ skin dạng data-driven:
  - `skin_loader.py`: đọc JSON skin, load sprite sheet, cắt frame theo `atlas_rect/src_rect`, build `entity.frames` và `entity.anim_sequences`.
  - `skin_screen.py`: UI chọn skin, preview, apply skin.

2.2.3. Điều phối lượt chơi trong `main.py`

Một lượt chơi trong `main.py` chạy đúng tinh thần sau:

1. UI nhận input (key hoặc click).
2. Nếu `current_turn == "player"` và tất cả actor idle: gọi `player.move()`.
3. Sau khi player move: gọi `check_special_tiles()` (key/trap), rồi `losing_check()`.
4. Nếu chưa chết: set `gamestate.pending_snapshot = True`, bắt đầu enemy turn (đi lần lượt).
5. Khi enemy đi xong và tất cả idle: chốt snapshot bằng `make_snapshot()` rồi push vào `gamestate.storedmove`.



Hình 2.1: Luồng điều phối lượt chơi trong `main.py`

2.3. Các class chính và cách tổ chức

2.3.1. `Cell`: mô hình ô lưới và tường/cổng

Mỗi `Cell(r,c)` là một node của map. Nó lưu trạng thái 4 cạnh:

`up, down, left, right`.

Quy ước trong code:

- `0`: không có tường (đi được).
- `1`: có tường (chặn).
- Riêng `down` có thể là **gate ngang**:
  - `2`: gate đóng
  - `3`: gate mở

Gate ngang được đồng bộ bởi `Gamestate.gates_h[(rh,c)]` và được sync về `Cell.down` khi toggle hoặc apply snapshot.

2.3.2. `Player`: input, animation, smooth slide

`Player` quản lý:

- Grid position: `row, col`, hướng `direction`.
- Lock di chuyển: `is_moving` (đang chạy animation thì không nhận input tiếp).
- Smooth slide: `render_dx/render_dy` và bộ đếm `move_timer/move_total`.
- Skin: `set_skin(skin_id)` gọi `skin_module.skin_loader.load_skin()` để build `frames` và sequence.

2.3.3. `Enemy`: AI theo mode, đi 1 hoặc 2 bước, có `uid`

`Enemy` có thêm vài thứ quan trọng:

- `uid`: định danh ổn định để encode BFS-state và để undo restore đúng con quái.
- `type`: `"white_mummy"`, `"red_mummy"`, `"red_scorpion"`.
- `pending_steps`: danh sách bước đã plan trước (phục vụ quái 2 bước/turn).
- `is_dead`: phục vụ cơ chế quái giết nhau và undo restore.

Cách triển khai 2 bước/turn:

- Trong `Enemy.move()`, game gọi hàm quyết định bước:
  - mode thường: dùng `new_enemy_position()` (Manhattan greedy)
  - impossible: dùng `impossible_mode_move()` (BFS đến player)
- Nếu không phải scorpion thì quái lấy `steps = 2`, plan ra `pending_steps`.
- `Enemy.update(dt)` sẽ chạy animation bước 1, xong thì tự gọi `_start_next_step()` để chạy bước 2.

2.3.4. `Gamestate`: state chung, items, undo history, solution, gate animation

`Gamestate` là nơi gom toàn bộ biến gameplay. Các field quan trọng đúng theo code:

- **Game flow**: `state`, `mode`, `chapter`, `level`, `gameover`, `result`.
- **Objective**: `goal_row`, `goal_col`.
- **Items**: `keys` (set), `traps` (set), `has_key`.
- **Gates**: `gates_h` (dict {(rh,c): bool}), `gate_anim_state`.
- **Undo/Restart**: `storedmove` (list snapshot), `initpos` (snapshot gốc), `pending_snapshot`.
- **BFS solution**: `solution` (list các hướng "up/down/left/right").
- **Difficulty**: `enemy_count`, `impossible_mode`.

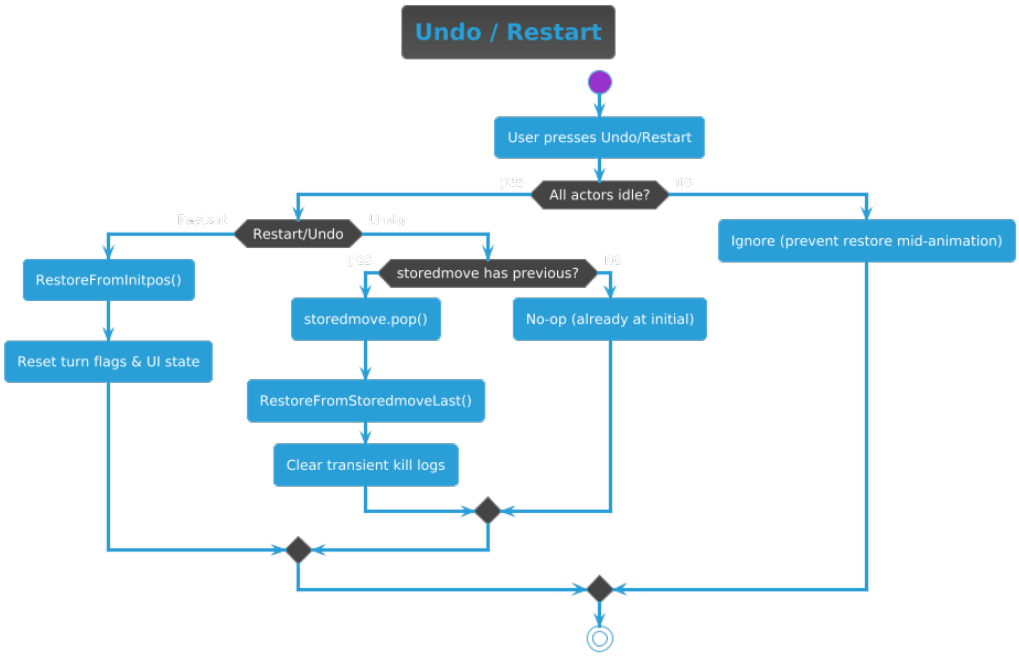
2.4. Undo và Restart

2.4.1. Tư tưởng triển khai

Undo/Restart trong game không cố **tua ngược animation**. Thay vào đó, game coi mỗi lượt là một **transaction**:

$$\text{Player action} \rightarrow \text{Enemy actions} \rightarrow \text{Commit snapshot}.$$

Undo chỉ việc lấy snapshot trước và `apply_snapshot()`, đảm bảo **đúng tuyệt đối**.



Hình 2.2: Quy trình Undo/Restart trong game

2.4.2. `make_snapshot()`: snapshot gồm những gì

Trong `module/module.py`, snapshot được build theo tuple, bám sát code:

- Player: `(player.row, player.col, player.direction, player.skin)`.
- Enemies: list các tuple:

$$(\texttt{uid}, \texttt{row}, \texttt{col}, \texttt{type}, \texttt{direction}, \texttt{pending_steps}, \texttt{is_dead}).$$

- Items:
  - `keys_state = set(gamestate.keys)`
  - `gates_state = dict(gamestate.gates_h)`
- Enemy kill log của lượt đó: `killed_state` (để undo khôi phục lại quái đã bị giết).

Chi tiết quan trọng:

- `killed_uids` được dùng để **không snapshot các con đã chết** theo đúng thời điểm.
- Snapshot được chốt **sau khi enemy đi xong**, nên storedmove phản ánh đúng **end-of-turn state**.

2.4.3. `apply_snapshot()`: restore state và reset các cờ movement

`apply_snapshot()` làm các việc:

1. Restore player pos/hướng/skin, reset `is_moving`, reset slide offset.
2. Rebuild list enemy từ `enemies_state` (tạo lại object `Enemy()`, `set_type()`, gán `uid`, `pending_steps`, `is_dead`).
3. Restore items: `gamestate.keys`, `gamestate.gates_h`.
4. Sync gate state về grid:

   `Cell.down = 3 (open) hoặc 2 (closed).`

5. Reset các cờ liên quan animation gate: rebuild `gamestate.gate_anim_state`.

2.4.4. Undo trong runtime (liên hệ với `main.py`)

Trong `main.py`, undo chỉ được cho phép khi **tất cả actor idle** để tránh restore giữa animation. Sau khi undo:

- clear `killed_uids` và `killed_this_turn` để tránh lọc nhầm enemy vừa được restore.

2.4.5. Restart

Restart quay về snapshot gốc:

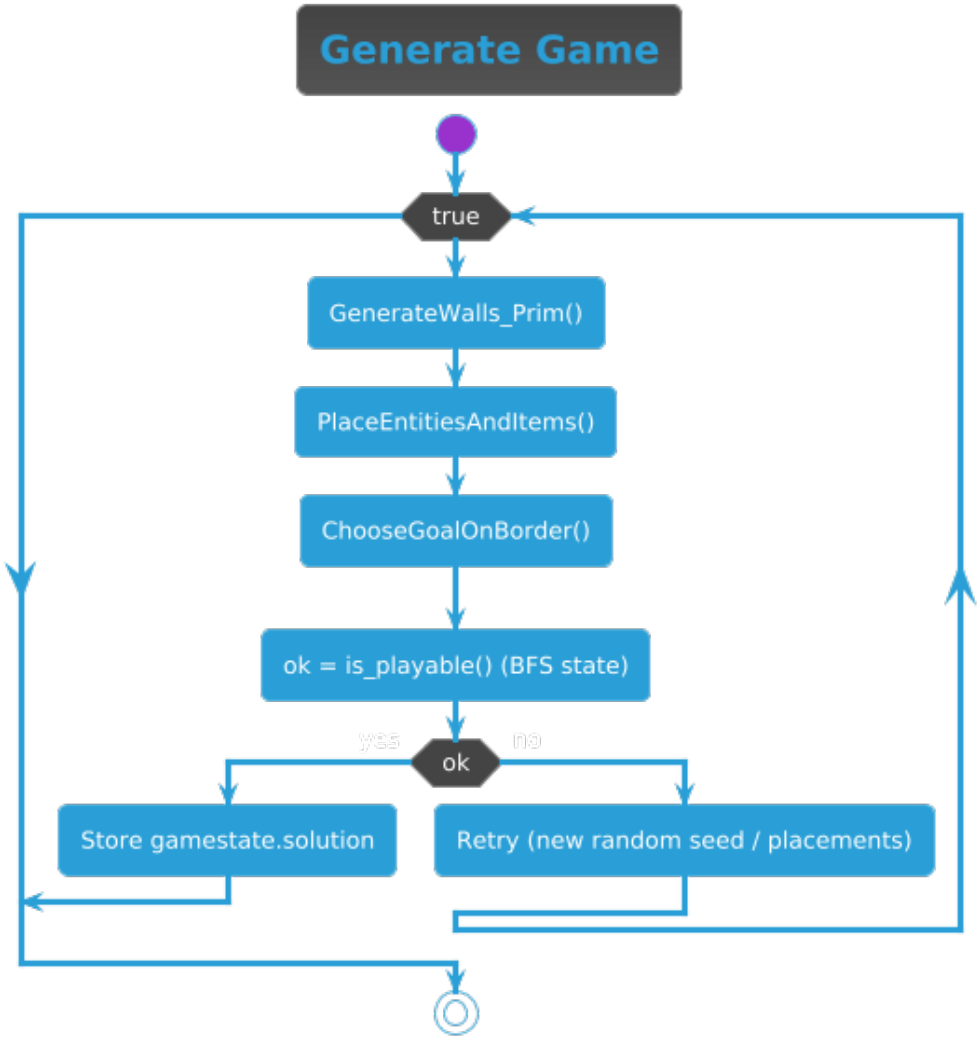
- Snapshot gốc được lưu trong `gamestate.initpos` ngay sau khi load/generate level.
- Restart thường sẽ:

  1. apply initpos,
  2. clear `storedmove` rồi push lại initpos,
  3. reset các cờ: `pending_snapshot, gameover, result`, reset turn index trong `main.py`.

2.5. Sinh màn chơi (Generate game)

2.5.1. Pipeline tổng trong `generate_game()`

`generate_game(grid, player, enemies, gamestate)` chạy theo pipeline:



Hình 2.3: Quy trình sinh màn chơi trong game

1. **Sinh tường/đường đi** bằng `generate_walls_pr()` (Prim).
2. **Random player/enemy/goal**:
   - player random một ô bất kỳ,
   - enemy random nhiều ô khác nhau (theo `gamestate.enemy_count`),
   - goal random trên **viền map** để tạo cảm giác "thoát ra ngoài".
3. **Kiểm tra solvable** bằng `is_playable()` (BFS state):
   - nếu fail thì sinh lại (loop),

- nếu pass thì lưu `gamestate.solution`.

2.5.2. Prim wall generate: `generate_walls_pr()`

Hàm `generate_walls_pr(grid, wall_density)` trong code làm 2 bước:

Bước 1: tạo maze liên thông (Prim kiểu randomized)

- Đóng tất cả tường nội bộ trước.
- Chọn 1 cell start ngẫu nhiên, đánh dấu visited.
- Duy trì danh sách frontier edges (các cạnh nối ra ngoài vùng visited).
- Mỗi lần chọn ngẫu nhiên 1 cạnh:
  - nếu cạnh nối visited $\leftrightarrow$ unvisited: mở tường giữa 2 cell, thêm cell mới vào visited, push frontier mới.
  - nếu không (nối visited-visited): bỏ qua.

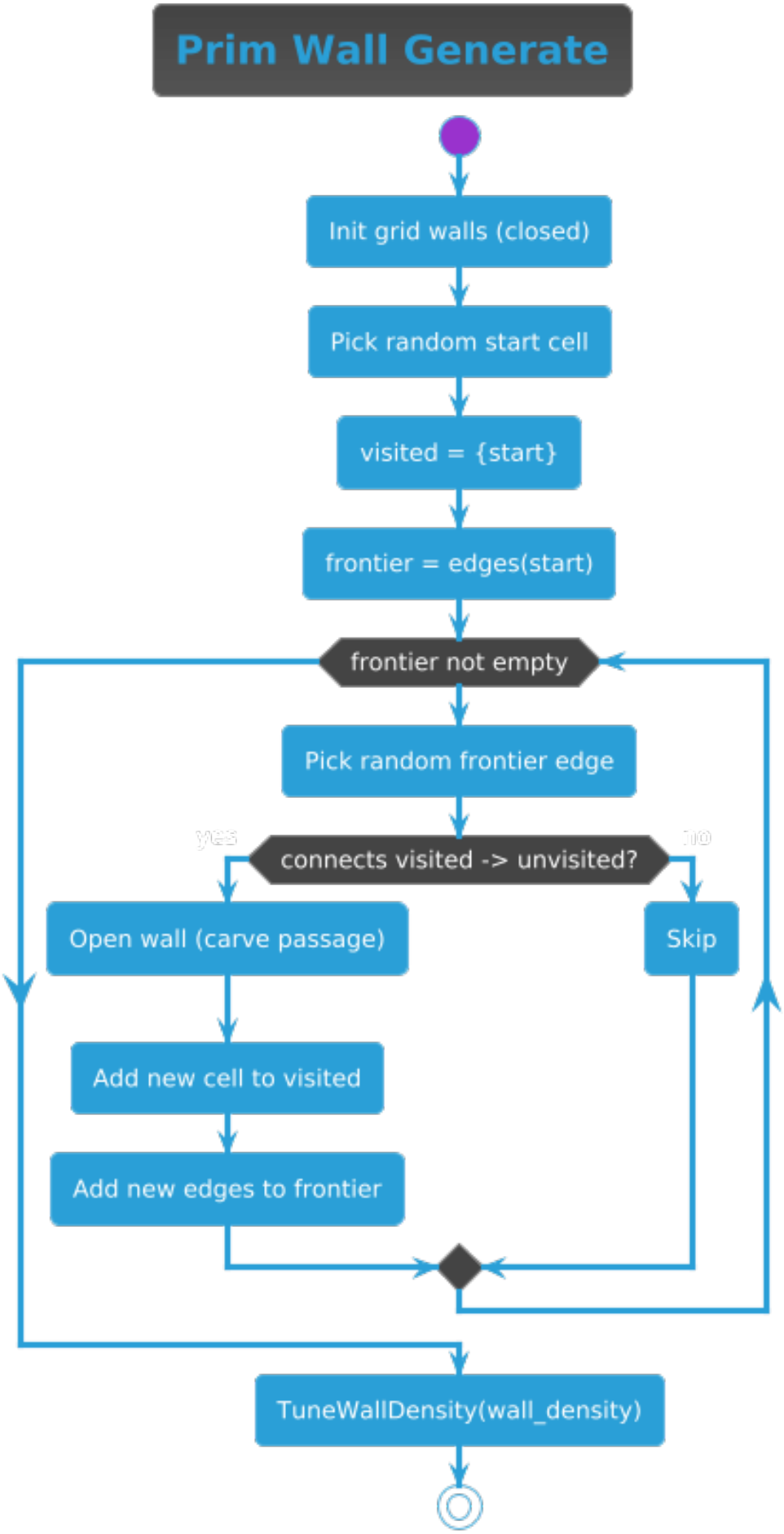
Kết quả của bước này là một spanning tree trên grid, nên **chắc chắn liên thông**.

Bước 2: mở thêm cạnh để đạt mật độ tường mong muốn Trong code, game tính:

$$\texttt{target_walls} = \lfloor \texttt{total_edges} \cdot \texttt{wall_density}/100 \rfloor.$$

Sau đó:

- liệt kê toàn bộ các cạnh đang đóng,
- mở thêm một số cạnh ngẫu nhiên để giảm số tường về gần `target_walls`.


Prim Wall Generate
Init grid walls (closed)
Pick random start cell
visited = {start}
frontier = edges(start)
frontier not empty
Pick random frontier edge
yes
connects visited -> unvisited?
no
Open wall (carve passage)
Skip
Add new cell to visited
Add new edges to frontier
TuneWallDensity(wall_density)

2.6. BFS path finding và kiểm tra solvable

2.6.1. BFS trong game không chỉ là BFS theo ô

Trong `module/module.py`, solvable được kiểm bằng `is_playable()`, thực chất gọi `path_finding()`. Điểm quan trọng: node của BFS là một **state**, không chỉ là `(row,col)`.

2.6.2. State mà BFS đang xét (đúng theo code)

State bao gồm:

- player position: `(pr, pc)`
- enemy snapshot: `encode_enemies(enemies_state)` (sort theo uid để ổn định)
- keys còn lại: `keys_state` (tuple sorted)
- gates state: `gates_state` (tuple sorted {(pos,is_open)})
- `has_key`

2.6.3. Transition trong BFS (mỗi bước tương ứng 1 lượt player)

Từ một state, BFS thử các action:

- 4 hướng di chuyển: up/down/left/right
- **skip** (player đứng yên) để mô phỏng input `K_SPACE` trong runtime

Mỗi action sẽ:

1. check hợp lệ theo tường/gate:
   - gate đóng `Cell.down == 2` thì không qua,
   - gate mở `Cell.down == 3` thì qua.
2. update key/gate:
   - nếu player đứng lên key tile $(r, c) \in$ `keys`: BFS giả lập `toggle_gates` bằng cách flip toàn bộ `gates_state`.

3. reject nếu player dẫm trap.

4. giả lập enemy turn (chi tiết ở dưới).

2.6.4. Giả lập enemy turn ngay trong BFS

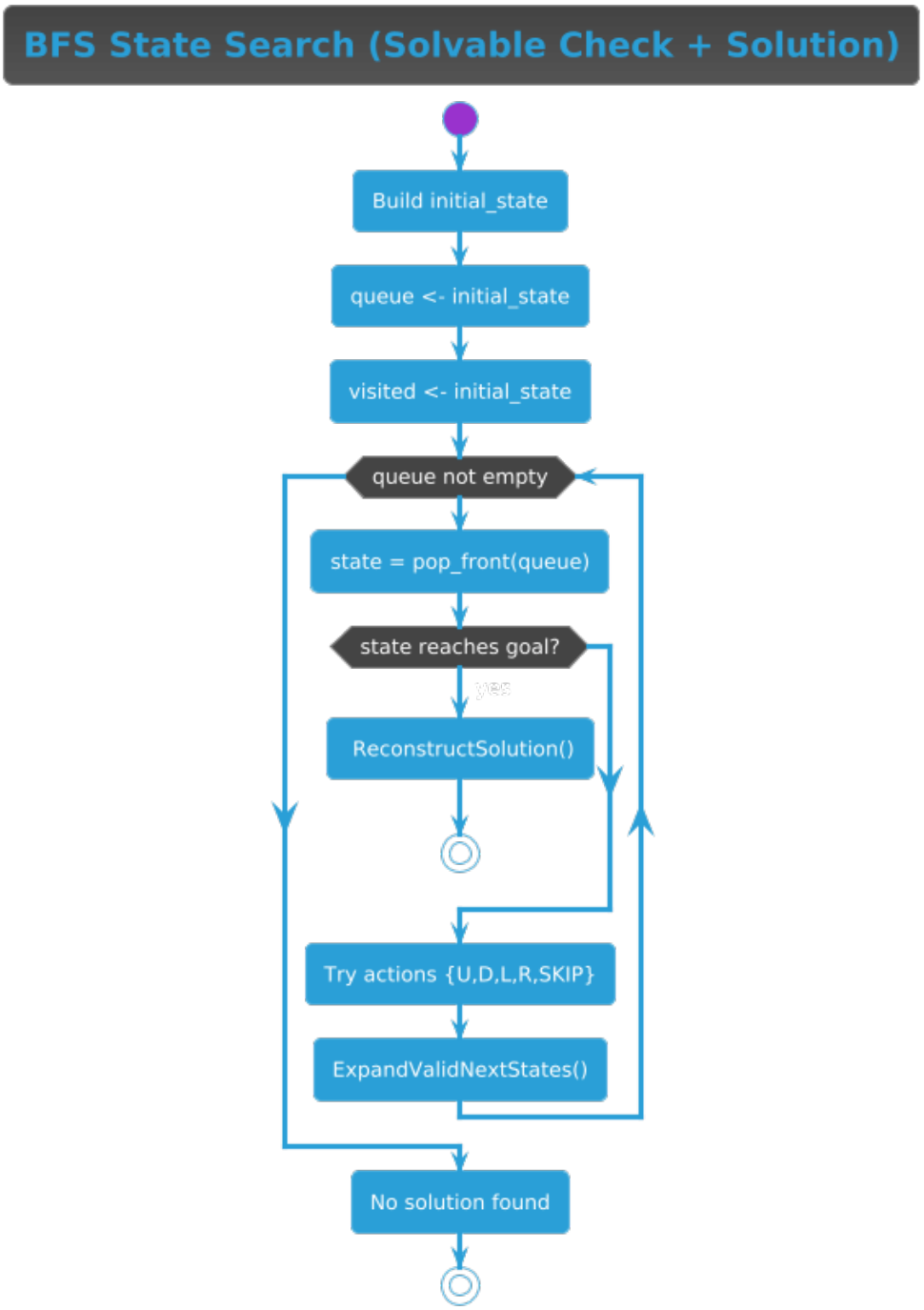
Đây là phần làm cho solvable check "đúng luật game", vì enemy cũng di chuyển theo lượt. Trong `path_finding()`, mỗi step BFS sẽ mô phỏng:

- Mỗi enemy có thể đi 1 hoặc 2 bước:

  `steps = 2 nếu type != red_scorpion, ngược lại 1.`

- Enemy chọn bước bằng:
  - `enemy_best_step_manhattan` nếu `impossible_mode = False`
  - `enemy_best_step_bfs` nếu `impossible_mode = True`
- Enemy né trap: không cho bước vào trap tile.
- Enemy-enemy collision:
  - nếu đạp lên vị trí enemy đứng yên: enemy đó bị coi là chết (đúng theo ý tưởng quái giết nhau),
  - nếu hai enemy cùng muốn vào một ô trong cùng lượt: enemy đi sau bị block (đứng yên).

Sau khi mô phỏng xong enemy, nếu có enemy trùng vị trí player thì state đó là **dead-end** (không enqueue).



Hình 2.5: Quy trình BFS trong game

2.6.5. Reconstruct solution

Khi BFS gặp goal $(pr, pc) = (\texttt{goal_row}, \texttt{goal_col})$, code reconstruct bằng parent map, và ghi `gamestate.solution` thành list hướng đi: `"up/down/left/right"`. Solution này được dùng trong **auto-play/hint**.

2.7. Hai thuật toán đường đi chính của quái

2.7.1. Manhattan greedy: `new_enemy_position()`

`new_enemy_position(player, enemy, grid)` là chiến lược nhanh:

- thử 4 hướng hợp lệ theo tường/gate,
- chọn hướng làm giảm khoảng cách Manhattan:

$$d = |r_e - r_p| + |c_e - c_p|.$$

Ưu điểm: nhẹ, tạo cảm giác "đuổi theo" nhưng vẫn bị maze làm lệch đường.

2.7.2. BFS (impossible mode): `impossible_mode_move()`

`impossible_mode_move(player, enemy, grid)` chạy BFS từ enemy tới player:

- node BFS là ô trên grid,
- xét `can_move_from` có tường/gate,
- lấy bước đầu tiên của đường đi ngắn nhất.

Nếu BFS không tìm được đường (hiếm nhưng vẫn có thể), code fallback sang Manhattan để tránh enemy đứng im.

2.7.3. Kết nối với cơ chế 2 bước/turn

Trong `Enemy.move()`:

- mỗi bước tạo một phần tử trong `pending_steps`,

- `Enemy.update(dt)` tự chạy hết `pending_steps` (step 1 xong thì start step 2).

Cách này giúp gameplay "turn-based" nhưng render vẫn mượt, và không phải nhét logic 2 bước vào main loop.

2.8. Tổ chức level JSON và save JSON

2.8.1. Level JSON (assets/map/*)

Level JSON đọc bởi `read_map_json(filepath)` với format chính:

- `"size": {"rows": R, "cols": C}`
- `"tiles"`: mảng string, mỗi ký tự là 1 tile:
  - `P`: player start
  - `W/R/S`: enemy (white_mummy / red_mummy / red_scorpion)
  - `K`: key
  - `T`: trap
  - `E`: exit (goal)
  - `.`: trống
- `"walls_v"`: tường dọc (mỗi row là string độ dài `cols+1`, ký tự `|` hoặc space).
- `"walls_h"`: tường ngang (mỗi row là string độ dài `cols`, ký tự: `-` wall, `=` gate đóng, `~` gate mở, space).

Khi load:

1. reset toàn bộ `Cell` về 0,
2. đọc `walls_v/walls_h` để set `left/right/up/down`,
3. nếu gặp `=` hoặc `~` thì fill `gamestate.gates_h[(r+1,c)]`,
4. đọc tiles để đặt player, tạo enemies, fill `keys/traps`, set goal.

Cuối cùng:

- set `gamestate.initpos = make_snapshot(...)`,
- clear và init `storedmove` với initpos,
- gọi `is_playable()` để build `gamestate.solution` cho adventure.

2.8.2. Save JSON (local profile, `saveload.py`)

Save được tổ chức theo cấu trúc:

- `player_info`: `username, password, is_guest, score, skin, owned_skins`.
- `classic`: block save riêng cho classic.
- `adventure`: block save riêng cho adventure.

Trong mỗi mode, save có thể chứa:

- `game_state`: (grid_size, chapter/level, player/enemies, goal, items, gates...).
- `map_data`: (tiles + walls_v/walls_h) để continue đúng map đang chơi.
- `move_history`: serialize từ `gamestate.storedmove` để continue + undo sau khi load.

Điểm quan trọng đúng theo hướng nhóm cần:

- Trong code, adventure cũng serialize theo cùng pipeline và vẫn trả về `map_data` (để continue không lệch map).

2.9. Sprite sheet, animation, và hệ thống skin

2.9.1. Quy ước sprite hướng và frame

Core animation của Player/Enemy dựa trên `entity.frames`:

```
frames["up"], frames["right"], frames["down"], frames["left"].
```

Mỗi list chứa nhiều frame, và index frame chạy theo `entity.anim_sequences["move"]`.

2.9.2. Cắt sprite sheet kiểu grid (legacy): `add_sprite_frames()`

Trong `module/module.py`, `add_sprite_frames(sheet, CELL_SIZE, frames)` cắt sheet theo lưới 4x5:

- row0 = up
- row1 = right
- row2 = down
- row3 = left

Mỗi ô được scale về `CELL_SIZE` và append vào `frames[dir]`.

2.9.3. Chạy animation + slide

Cả Player và Enemy đều dùng cùng cơ chế:

- Animation frame chạy bằng `anim_timer`, `anim_idx`.
- Slide chạy bằng `render_dx/render_dy` tiến dần về 0 theo thời gian.
- `is_moving` khóa input và khóa việc plan step mới cho tới khi slide xong.

CHƯƠNG 3

UI INTRODUCTION

3.1. Sơ lược giao diện game

3.1.1. Tổng quan trò chơi

Đồ án **Mummy Maze** là một ứng dụng trò chơi giải đố chiến thuật được phát triển dựa trên cảm hứng từ tựa game kinh điển của hãng PopCap. Trò chơi đặt người chơi vào bối cảnh các kim tự tháp Ai Cập huyền bí, nơi họ phải vận dụng khả năng tư duy logic để vượt qua các mê cung đầy rẫy nguy hiểm.

3.1.2. Công nghệ sử dụng

Đồ án được triển khai bằng ngôn ngữ lập trình `Python` kết hợp với thư viện `Pygame` và các tài nguyên đồ họa và Hoạt họa từ The Spriters Resource và My Abandonware.

- **Thư viện Pygame:** Là bộ công cụ mã nguồn mở hỗ trợ phát triển các ứng dụng đa phương tiện và trò chơi 2D. `Pygame` được lựa chọn nhờ khả năng xử lý đồ họa mượt mà, quản lý sự kiện bàn phím/chuột tối ưu và hệ thống âm thanh linh hoạt, giúp xây dựng cấu trúc trò chơi theo hướng đối tượng (OOP) một cách hiệu quả.
- **Tài nguyên đồ họa và Hoạt họa:** Toàn bộ hệ thống hình ảnh được trích xuất và tham khảo từ hai nguồn chính The Spriters Resource và My Abandonware:
  - *My Abandonware:* cung cấp các graphics và sounds *Mummy Maze Deluxe*.
  - *The Spriters Resource:* tạo các nhân vật người chơi dùng trong game
- **Logic điều khiển:** Hệ thống quản lý trạng thái để chuyển đổi giữa các màn hình như *Login, Register, Menu, Game Mode, Playing...*

3.2. Các chức năng của game:

3.2.1. Đồ họa và hoạt ảnh

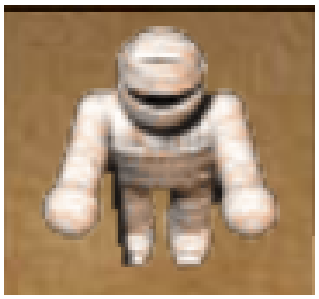
Dưới đây là danh sách các tài nguyên đồ họa chính được trích xuất và sử dụng trong trò chơi, đảm bảo tái hiện phong cách đặc trưng của phiên bản gốc The Spriters Resource và My Abandonware:

- **Nhân vật:** Có thể chọn nhân vật mặc định (nhà thàm hiểm) hoặc nhân vật đã mua được trong shop (có tất cả 11 loại nhân vật).

Nhà thám hiểm

Nhân vật từ shop

- **Quái vật:** Tấn công người chơi (Có thể nhiều hơn 1 loại quái vật trong 1 màn chơi).



Xác ướp trắng

Xác ướp đỏ

Bọ cạp

Hình 3.1: Các loại quái vật trong mê cung.

- **Chướng ngại vật và Vật phẩm:** Tường và hàng rào cản trở việc di chuyển. Bẫy hình đầu lâu có thể giết người chơi. Chìa khóa có thể dùng để mở hoặc đóng hàng rào.

Tường đá

Hàng rào

Chìa khóa

Bẫy/Đầu lâu

Hình 3.2: Các chướng ngại vật và vật phẩm.

3.2.2. Hệ thống Quản lý tài khoản:

Đăng ký tài khoản

Nếu chưa có tài khoản, người chơi có thể tạo tài khoản mới ở đây

Hình 3.3: Màn hình Đăng ký tài khoản mới.

Đăng nhập

Khi đã có tài khoản, người chơi sẽ nhập thông tin tài khoản tại đây để truy cập vào trò chơi.

Hình 3.4: Màn hình Đăng nhập.

Chức năng Chơi với tư cách khách (Play as Guest)

Đây là tính năng hỗ trợ người dùng trải nghiệm nhanh trò chơi mà không cần thông qua các bước khởi tạo tài khoản.

- **Mô tả:** Người chơi chỉ cần nhấn vào nút "Play as Guest" và chọn tài khoản guest có sẵn thì sẽ vào được game.
  - Hệ thống sẽ cấp một định danh tạm thời cho phiên chơi đó.
  - *Hạn chế:* Kết quả chơi, điểm số hoặc tiến trình vượt ải sẽ không được lưu lại bền vững trong cơ sở dữ liệu sau khi xóa ứng dụng.

Hình 3.5: Chế độ khách.

3.2.3. Giao diện Menu chính

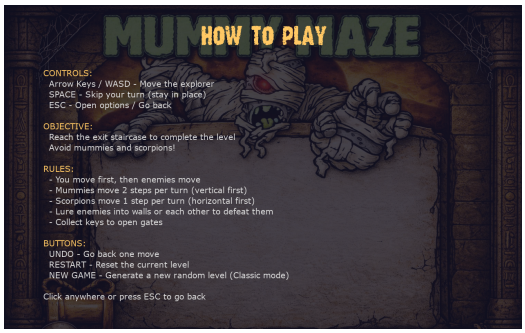
Sau khi truy cập thành công vào trò chơi, người chơi sẽ được đưa đến màn hình Menu chính. Đây là trung tâm điều hướng của toàn bộ ứng dụng.

Hình 3.6: Giao diện Menu chính.

- **Play :** Chuyển đến màn hình lựa chọn chế độ chơi (Game Mode).

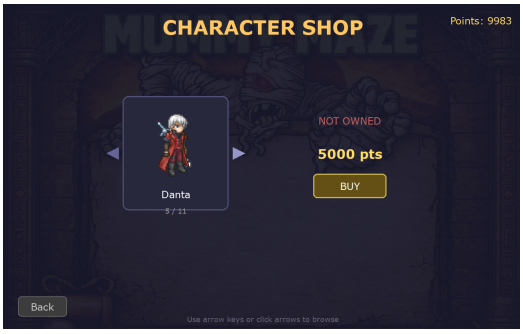
Hình 3.7: Các chế độ game.

- **Tutorial :** Cung cấp các chỉ dẫn trực quan về cơ chế di chuyển theo lượt và các quy tắc đặc thù trong mê cung, giúp người chơi mới dễ dàng tiếp cận trò chơi.



Hình 3.8: Hướng dẫn.

- **Character :** Khu vực shop nhân vật. Tại đây, người chơi có thể chọn nhân vật là nhà thám hiểm mặc định hoặc các nhân vật đã mua bằng điểm.



Hình 3.9: Character shop

- **LeaderBoard :** Hiển thị danh sách các "nhà thám hiểm" xuất sắc nhất. Chức năng này thúc đẩy tính cạnh tranh giữa các người chơi thông qua việc lưu trữ và so sánh điểm số từ cơ sở dữ liệu.

Hình 3.10: Bảng xếp hạng.

3.2.4. Lựa chọn chế độ chơi

Sau khi người chơi nhấn chọn **Play** tại Menu chính, hệ thống sẽ chuyển sang giao diện lựa chọn chế độ chơi. Việc phân chia này giúp người chơi có những trải nghiệm khác nhau về mặt chiến thuật và không gian:

- **Classic Mode:** Trong chế độ này, người chơi sẽ đối mặt với các thử thách được phân cấp rõ rệt về độ khó. Hệ thống cung cấp 4 mức độ để người chơi lựa chọn:
  - **Easy (Mê cung 6x6):** Dành cho người mới bắt đầu để làm quen với quy luật di chuyển.
  - **Medium (Mê cung 8x8):** Tăng số lượng vật cản và vị trí xuất phát của kẻ thù khó khăn hơn.
  - **Hard (Mê cung 10x10):** Kết hợp nhiều loại xác ướp và bẫy trong một diện tích hẹp.
  - **Impossible (Không tưởng):** Thử thách cực hạn, yêu cầu người chơi phải điều hướng tối ưu để không bị dồn vào đường cùng.

Hình 3.11: Các mức độ thử thách trong chế độ Classic.

- **Adventure Mode:**
  - Mở rộng trải nghiệm với các bản đồ có cấu trúc phức tạp và đa dạng về môi trường (Hầm mộ đá, Kim tự tháp vàng...).
  - Bản đồ sẽ gồm các levels với mỗi level chơi sẽ gồm 15 màn. Mỗi màn chơi xong, sẽ đánh dấu vào 1 ô trong 15 ô của hình tam giác.

Hình 3.12: Giao diện Bản đồ thế giới trong chế độ Adventure.

3.2.5. Giao diện Màn hình chơi

Giao diện chơi game được thiết kế đồng nhất về mặt hình họa nhưng có sự phân hóa về chức năng nút bấm để phù hợp với từng mục tiêu chơi.

Giao diện Chế độ Classic

Trong chế độ Classic, dải menu bên trái cung cấp các công cụ tối ưu cho việc giải đố liên tục. Đặc biệt là nút **NEW GAME** giúp người chơi nhanh chóng chuyển sang các thử thách mới.

Hình 3.13: Giao diện màn hình chơi chế độ Classic.

Giao diện Chế độ Adventure

Đối với chế độ Adventure, hệ thống điều khiển được thay đổi để phục vụ tính chất khám phá. Nút **NEW GAME** được thay thế bằng biểu tượng hình kim tự tháp hiển thị tiến trình game.

Hình 3.14: Giao diện màn hình chơi chế độ Adventure.

So sánh các nút chức năng chính

- **RESTART/UNDO:** Có mặt ở cả hai chế độ để hỗ trợ người chơi sửa sai trong các bước đi logic.
- **Phím di chuyển ảo:** Xuất hiện ngay dưới chân nhân vật (mũi tên xanh) giúp người chơi điều khiển trực quan bằng chuột hoặc cảm ứng.
- **Khu vực hiển thị:** Cả hai đều hiển thị rõ ràng vị trí nhân vật, xác ướp và cửa thoát hiểmnằm ở góc dưới bên phải lưới.
- **OPTIONS:** Dùng để hiển thị màn hình điều chỉnh độ lớn âm thanh nền, âm thanh hiệu ứng, tốc độ di chuyển nhân vật.

- **EXIT:** Khi bấm vào, người chơi sẽ thoát về màn hình menu khi bấm "YES" và lưu lại tiến trình của người chơi.

Hình 3.15: OPTIONS

Hình 3.16: EXIT

3.2.6. Giao diện khi thất bại

Các trạng thái khi nhân vật bị chết

Do xác ướp trắng

Do xác ướp đỏ

Do bọ cạp

Do rơi vào bẫy

Hình 3.17: Minh họa các trạng thái kết thúc trò chơi.

Sau khi chết, màn hình game over xuất hiện từ dưới lên.

Hình 3.18: Giao diện thông báo khi người chơi chết.

Giao diện này cung cấp 4 chức năng điều hướng chính:

- **TRY AGAIN:** Tải lại màn chơi hiện tại.
- **UNDO MOVE:** Hoàn tác bước đi sai lầm cuối cùng để người chơi có thể thử lại hướng di chuyển khác.
- **ABANDON HOPE:** Chấp nhận kết thúc phiên chơi hiện tại và coi hướng dẫn giải.
- **BACK TO MAIN MENU:** Quay trở về giao diện Menu chính của ứng dụng.