



PROJET LONG

Bonjours



Pokémon Donjon Mystère

- Concevoir un jeu codé en langage fonctionnel.
- Jeux souvent développés en langage objet (Java, C#, ...).
- OCaml utilise des types algébriques, immuabilité et fonctions pures.
- Une approche plus rigoureuse, modulaire et sûre.





Pokémon Donjon Mystère

GAMEPLAY

- Le joueur s'aventure dans un donjon généré de manière procédurale, combat des ennemis, collecte des trésors, évite les pièges et avance à travers des niveaux de difficulté de plus en plus élevés.

Level: 11 HP: 105/105



Calendrier

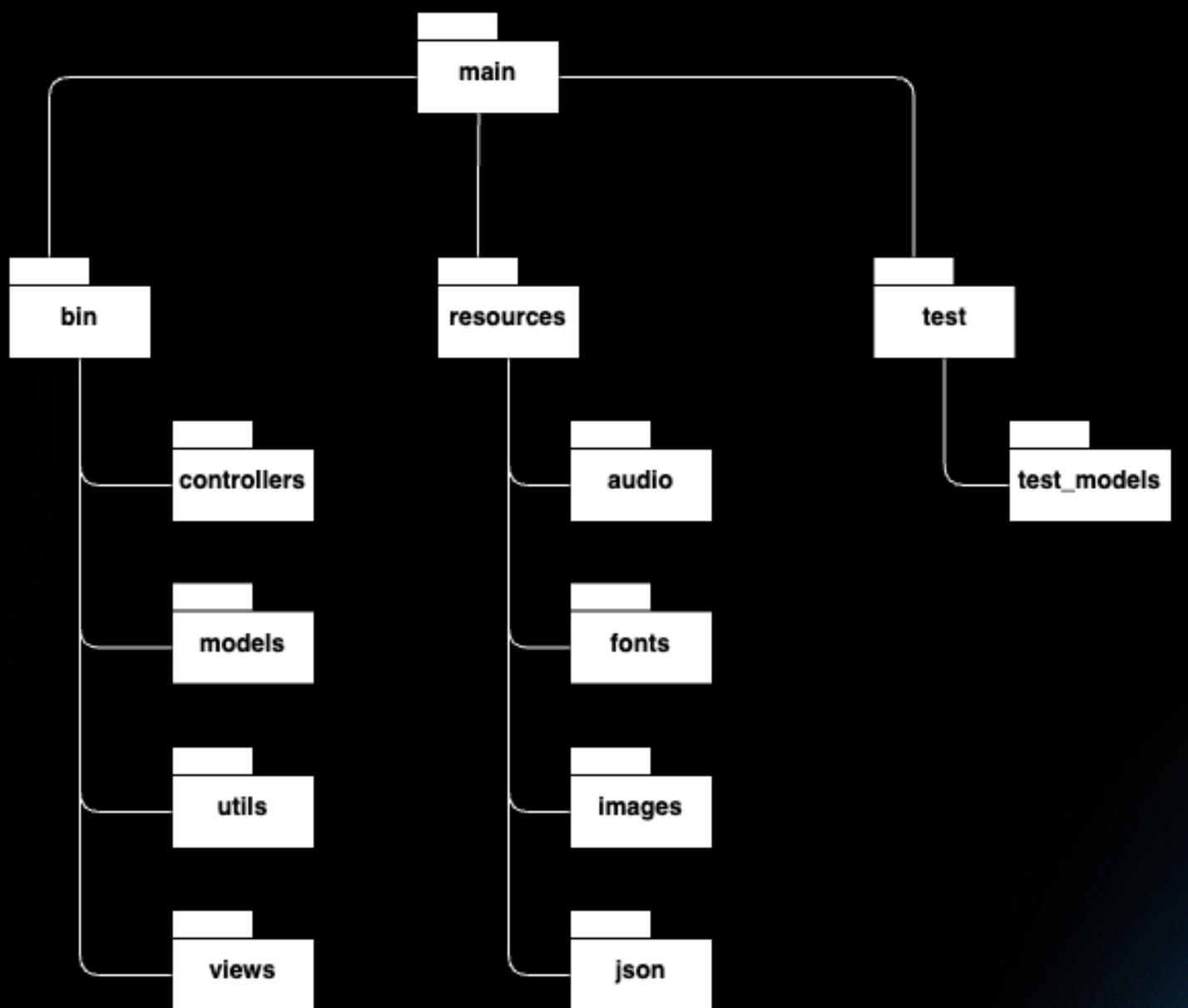
Semaine																							
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	
Tâche 0.1	Tâche 0.2																						
		Tâche 1.1					Tâche 1.2		Tâche 1.3				Tâche 1.4		Tâche 1.5		Tâche 1.6		Tâche 1.7				
		Tâche 2.1		Tâche 2.2	Tâche 2.3		Tâche 2.4		Tâche 2.5				Tâche 2.6	Tâche 2.7				Tâche 2.8		Tâche 2.9			
			Tâche 3.1			Tâche 3.2		Tâche 3.3				Tâche 3.4			Tâche 3.5	Tâche 3.6	Tâche 3.7		Tâche 3.8			Tâche 3.9	

- Tache 1.x : Génération de la carte.
- Tache 2.x : Affichage du jeu.
- Tache 3.x : Logique du jeu.





Architecture





Structure principaux

```
type game_state = {
    map_state: map;
    player_state: pokemon;
    enemies_state: pokemon list;
    loots_state: loot list;
    traps_and_grounds_state: trap_and_ground list;
    msgs_state: string list;
}
```

```
type game_textures = {
    tiles_tex: Raylib.Texture.t list;
    entities_tex: Raylib.Texture.t list;
    items_tex: Raylib.Texture.t list;
    bag_tex: bag_textures;
    shadow_cast_tex: Raylib.Texture.t list;
    traps_and_grounds_tex: Raylib.Texture.t list;
    attack_msg_tex: Raylib.Texture.t option;
}
```

```
type screenState =
| Intro
| Select
| Select_New
| Select_Other
| NewGame
| ChoosePokemon
| Game
```





Testes Unitaires

TESTES

- Tests unitaires pour les fonctions critiques :
 - Génération de carte : vérifier la cohérence des salles
 - A* renvoie bien un chemin valide
- Tests réalisés avec des modules de test OCaml - OUnit



```
let test_set_entity_target _ =
  let entity = empty_entity () in
  Printf.printf "Testing set_entity_target :\n";

  (* Test de la direction Up *)
  Printf.printf "- Testing Up direction\n";
  let updated_entity = set_entity_target_with_direction Up entity in
  let (target_x, target_y) = get_entity_target updated_entity in
  assert_equal 0.0 target_x;
  assert_equal (-1.0) target_y;

  (* Test de la direction Down *)
  Printf.printf "- Testing Down direction\n";
  let updated_entity = set_entity_target_with_direction Down entity in
  let (target_x, target_y) = get_entity_target updated_entity in
  assert_equal 0.0 target_x;
  assert_equal 1.0 target_y;

  (* Test de la direction Left *)
  Printf.printf "- Testing Left direction\n";
  let updated_entity = set_entity_target_with_direction Left entity in
  let (target_x, target_y) = get_entity_target updated_entity in
  assert_equal (-1.0) target_x;
  assert_equal 0.0 target_y;
```



Difficulté du code

PARTIE

- Génération procédurale logique.
 - Automate cellulaire
 - Prim
 - Voronoï
 - Probabilité
- Système de FOV
 - Shadowcasting
- IA des ennemis



```
    resp_iter = self.stub.GetStatuses(stub_id)
    statuses = {}
    for data in resp_iter:
        status = Status(
            status_id=data.id, name=data.name,
            type=data.type, value=data.value,
            min_value=data.min_value, max_value=data.max_value)
```

```
let flood_fill (tiles: tile list) (visited: (int * int) list ref) (x: int) (y: int) : int =
  let directions = [(1, 0); (-1, 0); (0, 1); (0, -1)] in
  let rec dfs (stack : (int * int) list) (zone_size : int) : int =
    match stack with
    | [] -> zone_size
    | (cx, cy) :: rest ->
      if not (List.exists (fun (vx, vy) -> vx = cx && vy = cy) !visited) then (
        visited := (cx, cy) :: !visited;
        let new_stack = List.fold_left (fun acc (dx, dy) ->
          let nx, ny = (cx + dx, cy + dy) in
          if List.exists (fun tile -> tile.x = nx && tile.y = ny && tile.texture_id = 1) tiles then
            (nx, ny) :: acc
          else
            acc
        ) rest directions in
        dfs new_stack (zone_size + 1)
      ) else
        dfs rest zone_size
  in
  dfs [(x, y)] 0
```



CONCLUSION

Merci