Soutenance de projet : Goldigger

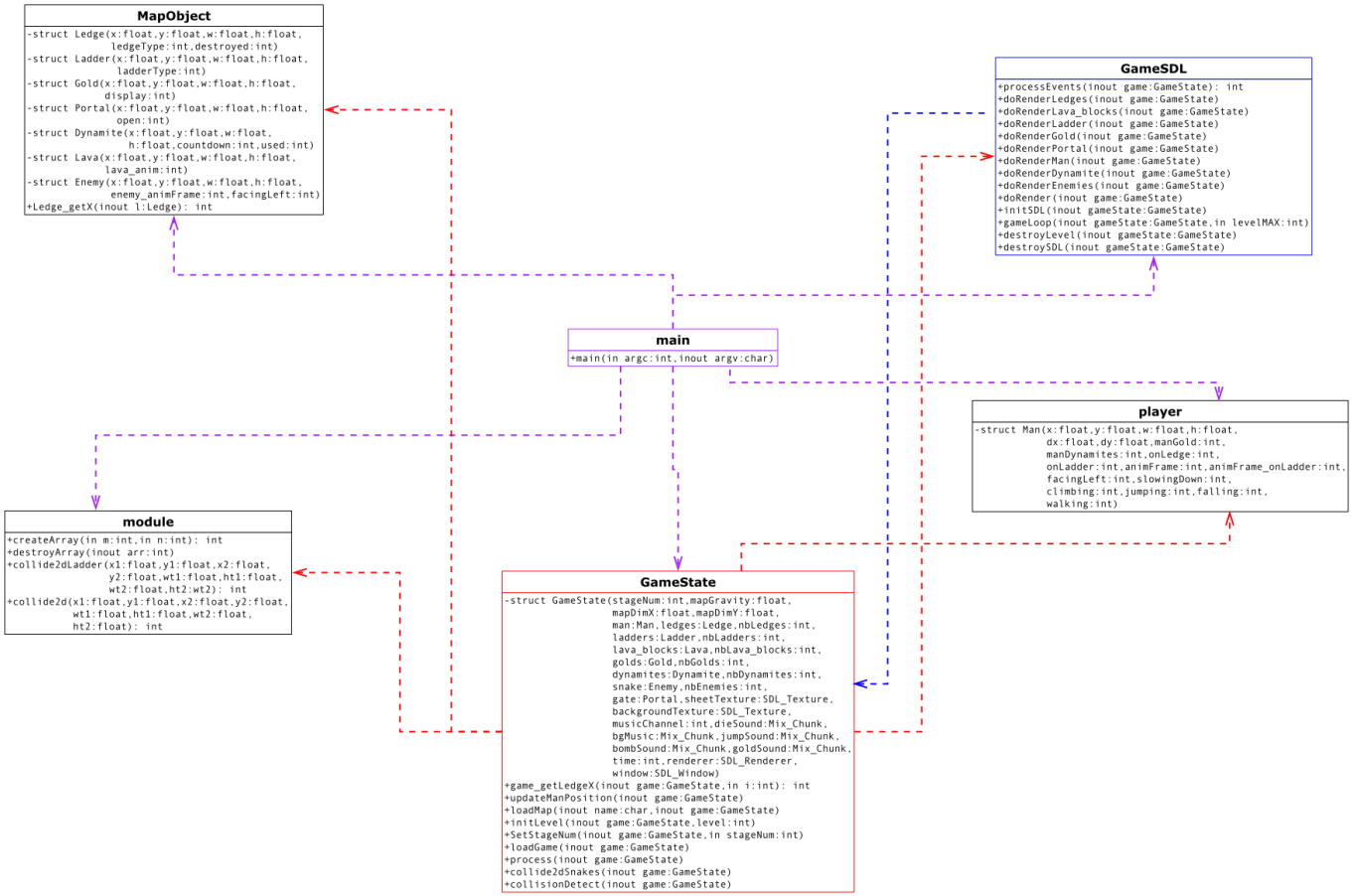
Alexandre HASSLER, Eloi POYET, Idir DJEMAOUNE

UCBL Lyon 1 - Mathématique Informatique

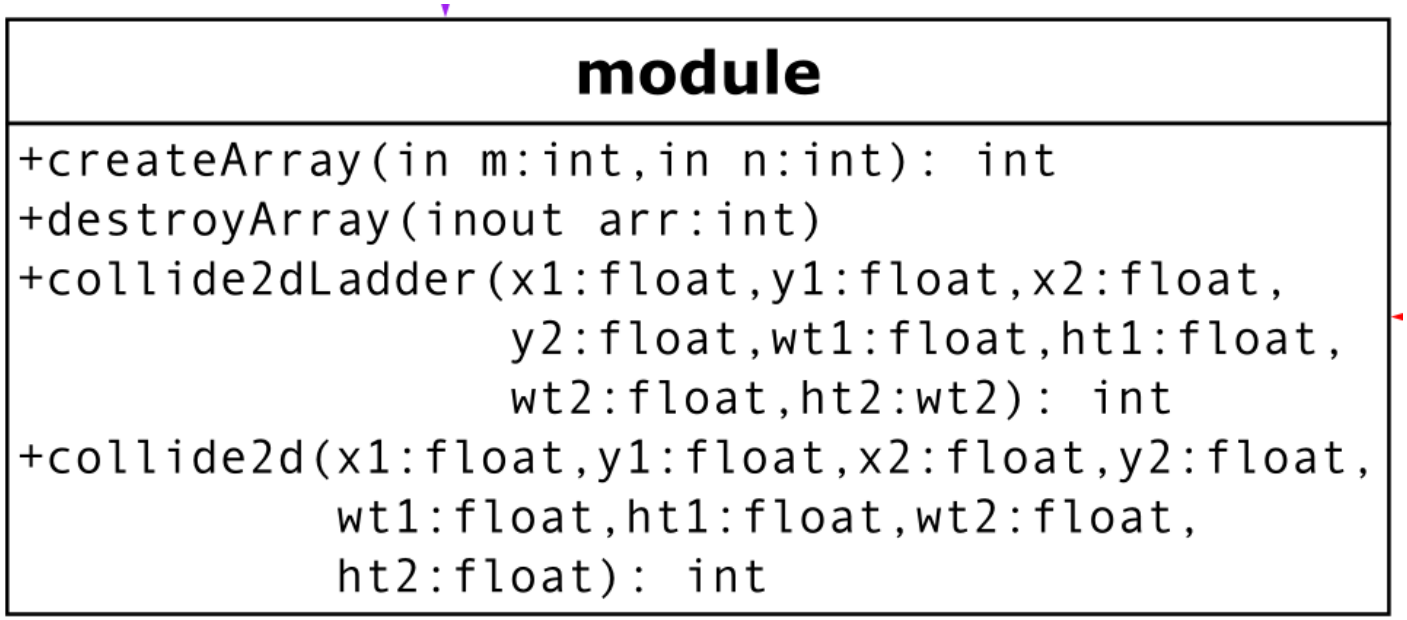
May 23, 2016


Game Window

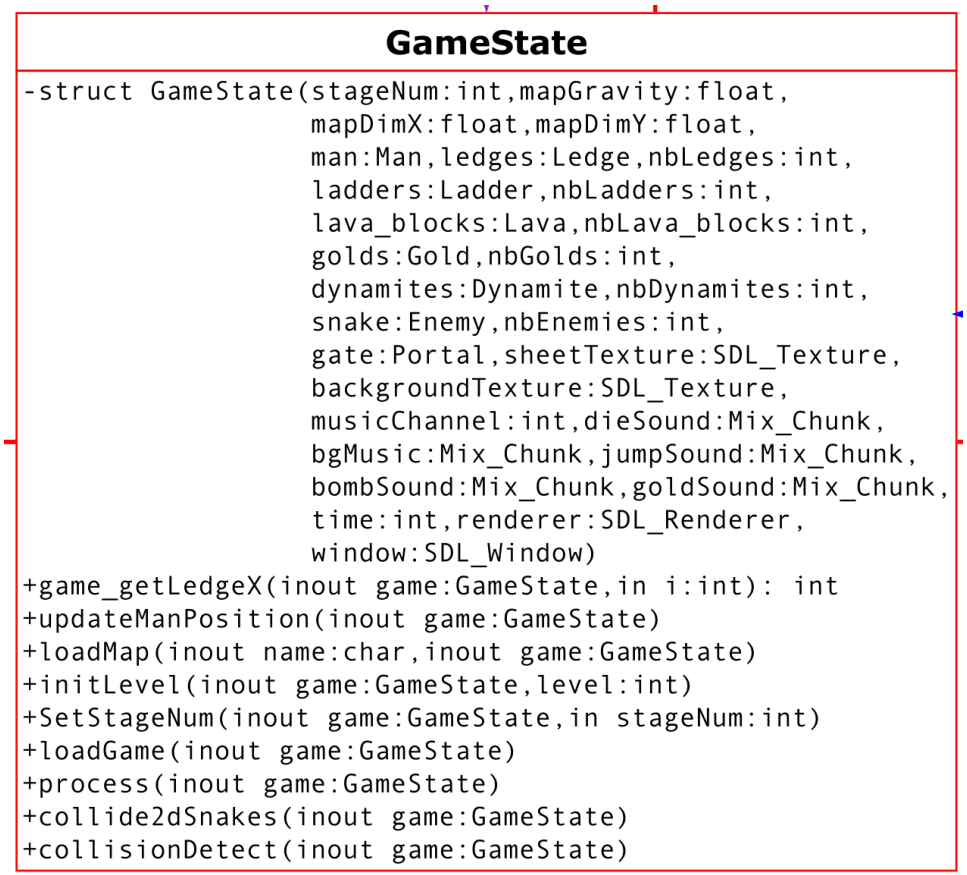
- moteur graphique 2D, implémentation **Tile-based** (smooth)
- **collision** déterminée avec **tilemap** et système **AABB** (Axis-Aligned Bounding Box)
- liberté de mouvement $\Longrightarrow$ déplacements élémentaires $\mathrm{d}x, \mathrm{d}y$
- **flags** $\Longleftrightarrow$ événements du clavier
- smooth jump, double jump, **gravity**, décélération
- autres mots clés : hitbox, **sprite caré**


MapObject
-struct Ledge(x:float,y:float,w:float,h:float,
ledgeType:int,destroyed:int)
-struct Ladder(x:float,y:float,w:float,h:float,
ladderType:int)
-struct Gold(x:float,y:float,w:float,h:float,
display:int)
-struct Portal(x:float,y:float,w:float,h:float,
open:int)
-struct Dynamite(x:float,y:float,w:float,
h:float,countdown:int,used:int)
-struct Lava(x:float,y:float,w:float,h:float,
lava_anim:int)
-struct Enemy(x:float,y:float,w:float,h:float,
enemy_animFrame:int,facingLeft:int)
+Ledge_getX(inout l:Ledge): int
GameSDL
+processEvents(inout game:GameState): int
+doRenderLedges(inout game:GameState)
+doRenderLava_blocks(inout game:GameState)
+doRenderLadder(inout game:GameState)
+doRenderGold(inout game:GameState)
+doRenderPortal(inout game:GameState)
+doRenderMan(inout game:GameState)
+doRenderDynamite(inout game:GameState)
+doRenderEnemies(inout game:GameState)
+doRender(inout game:GameState)
+initSDL(inout gameState:GameState)
+gameLoop(inout gameState:GameState,in levelMAX:int)
+destroyLevel(inout gameState:GameState)
+destroySDL(inout gameState:GameState)
main
+main(in argc:int,inout argv:char)
player
-struct Man(x:float,y:float,w:float,h:float,
dx:float,dy:float,manGold:int,
manDynamites:int,onLedge:int,
onLadder:int,animFrame:int,animFrame_onLadder:int,
facingLeft:int,slowingDown:int,
climbing:int,jumping:int,falling:int,
walking:int)
module
+createArray(in m:int,in n:int): int
+destroyArray(inout arr:int)
+collide2dLadder(x1:float,y1:float,x2:float,
y2:float,wt1:float,ht1:float,
wt2:float,ht2:wt2): int
+collide2d(x1:float,y1:float,x2:float,y2:float,
wt1:float,ht1:float,wt2:float,
ht2:float): int
GameState
-struct GameState(stageNum:int,mapGravity:float,
mapDimX:float,mapDimY:float,
man:Man,ledges:Ledge,nbLedges:int,
ladders:Ladder,nbLadders:int,
lava_blocks:Lava,nbLava_blocks:int,
golds:Gold,nbGolds:int,
dynamites:Dynamite,nbDynamites:int,
snake:Enemy,nbEnemies:int,
gate:Portal,sheetTexture:SDL_Texture,
backgroundTexture:SDL_Texture,
musicChannel:int,dieSound:Mix_Chunk,
bgMusic:Mix_Chunk,jumpSound:Mix_Chunk,
bombSound:Mix_Chunk,goldSound:Mix_Chunk,
time:int,renderer:SDL_Renderer,
window:SDL_Window)
+game_getLedgeX(inout game:GameState,in i:int): int
+updateManPosition(inout game:GameState)
+loadMap(inout name:char,inout game:GameState)
+initLevel(inout game:GameState,level:int)
+SetStageNum(inout game:GameState,in stageNum:int)
+loadGame(inout game:GameState)
+process(inout game:GameState)
+collide2dSnakes(inout game:GameState)
+collisionDetect(inout game:GameState)

- mauvaise méthode de conception $\Longrightarrow$ **structure de structure** et peu de **mutateurs** $\Longrightarrow$ apprendre de ses erreurs
- les points positifs : une certaine **lisibilité** du code, la structure GameState fonctionne comme un **agrégateur de structures** $\Longrightarrow$ une seule **variable formelle**
- les points négatifs : respecte peu la modularité


module
+createArray(in m:int,in n:int): int
+destroyArray(inout arr:int)
+collide2dLadder(x1:float,y1:float,x2:float,
y2:float,wt1:float,ht1:float,
wt2:float,ht2:wt2): int
+collide2d(x1:float,y1:float,x2:float,y2:float,
wt1:float,ht1:float,wt2:float,
ht2:float): int

```
1 int collide2dLadder(float x1, float y1, float x2, ↩
      float y2, float wt1, float ht1, float wt2, float ↩
      ht2)
2 {
3   return (((x1 + wt1/2 > x2) && (x1 + wt1/2 < x2 + ↩
        wt2)) && ((y1 <= y2 + ht2) && (y1 + ht1 >= y2))↩
        );
4 } // test de collision avec les echelles
5
6 int collide2d(float x1, float y1, float x2, float y2,↩
       float wt1, float ht1, float wt2, float ht2)
7 {
8   return (!((x1 >= (x2+wt2/3)) || (x2 >= (x1+wt1/3)) ↩
        || (y1 >= (y2+ht2/3)) || (y2 >= (y1+ht1/3))));
9 } // test d'intersection avec 2 blocs
```


GameState
-struct GameState(stageNum:int,mapGravity:float,
mapDimX:float,mapDimY:float,
man:Man,ledges:Ledge,nbLedges:int,
ladders:Ladder,nbLadders:int,
lava_blocks:Lava,nbLava_blocks:int,
golds:Gold,nbGolds:int,
dynamites:Dynamite,nbDynamites:int,
snake:Enemy,nbEnemies:int,
gate:Portal,sheetTexture:SDL_Texture,
backgroundTexture:SDL_Texture,
musicChannel:int,dieSound:Mix_Chunk,
bgMusic:Mix_Chunk,jumpSound:Mix_Chunk,
bombSound:Mix_Chunk,goldSound:Mix_Chunk,
time:int,renderer:SDL_Renderer,
window:SDL_Window)
+game_getLedgeX(inout game:GameState,in i:int): int
+updateManPosition(inout game:GameState)
+loadMap(inout name:char,inout game:GameState)
+initLevel(inout game:GameState,level:int)
+SetStageNum(inout game:GameState,in stageNum:int)
+loadGame(inout game:GameState)
+process(inout game:GameState)
+collide2dSnakes(inout game:GameState)
+collisionDetect(inout game:GameState)

```
1  float bx = game_getLedgeX(game,i), by = game->ledges↩
       [i].y, bw = game->ledges[i].w, bh = game->ledges↩
       [i].h;
2  if(mx+mw/2 > bx && mx+mw/2<bx+bw)
3     { // est ce quon se cogne la tete ?
4       if(my < by+bh && my > by && game->man.dy < 0)
5       {
6         game->man.y = by+bh; // correction de la ↩
              position
7         my = by+bh;
8
9         game->man.dy = 0;
10        game->man.onLedge = 1; // si on se cogne on ↩
              retombe
11      }
12    }
```

```
1   Man *man = &game->man;
2   man->x += man->dx;
3   man->y += man->dy; // on recoit les deplacements ↩
        elementaires de Eventprocess
4
5  if(!game->man.climbing)
6  {
7   if(man->onLedge && !man->slowingDown) // test des ↩
        flags
8   {
9     if(game->time % 2 == 0)
10    {
11      //sheet
12      man->animFrame %=11; // 12 sprites d'animation
13      man->animFrame++;
14    }
15  }
```

```
1  int** createArray(int m, int n)
2  {
3      int* values = calloc(m*n, sizeof(int));
4      int** rows = malloc(n*sizeof(int*));
5      for (int i=0; i<n; ++i)
6      {
7          rows[i] = values + i*m;
8      }
9      return rows;
10 }
11
12 void destroyArray(int** arr)
13 {
14     free(*arr);
15     free(arr);
16 }
```


Game Window