Puissance 4 SDL - Séparation IHM LM
I3 Algorithmique

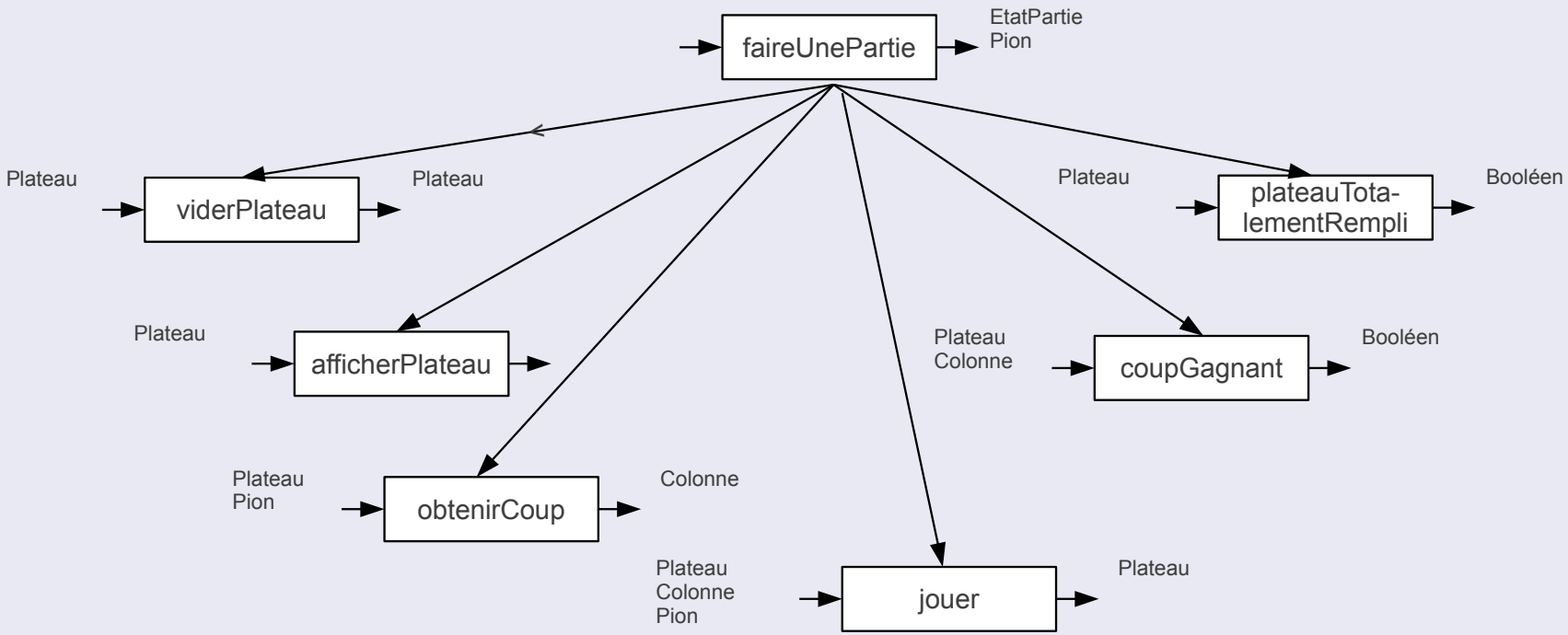
Nicolas Delestre

Plan

Architecture des Systèmes d'Information

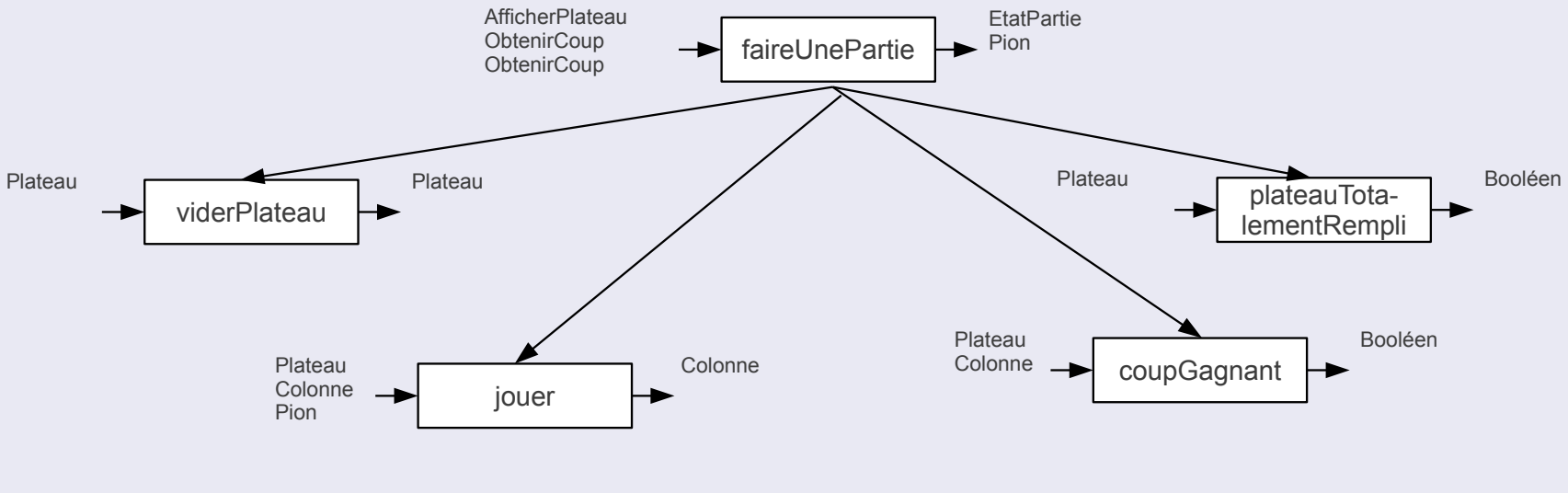
Analyse descendante 1 / 2

Analyse Descendante - avant



Analyse descendante 2 / 2

Analyse Descendante - maintenant



Conception préliminaire

Nouveaux types

Type AfficherPlateau = **procédure (E** p : Plateau**)**
Type ObtenirCoup = **fonction (**p : Plateau, joueur : Pion**) :** Colonne
⌊**précondition(s)** non plateauTotalementRempli(p)

Nouvelle signature

procédure faireUnePartie **(E** afficher : AfficherPlateau, obtenirCoupJaune, obtenirCoupRouge :ObtenirCoup, **S** etatPartie : EtatPartie, quiAGagne : Pion**)**

Conception détaillée 1 / 2

faireUnePartie - avant

procédure faireUnePartie **(S** etatPartie : EtatPartie, quiAGagne : Pion**)**

Déclaration lePlateau : Plateau, joueurCourant : Pion, coup : Colonne

debut

vider(lePlateau)
joueurCourant ← pionJaune
afficher(lePlateau)
repeter
coup ← obtenirCoup(lePlateau,joueurCourant)
jouer(lePlateau,coup,joueurCourant)
afficher(lePlateau)
joueurCourant ← autreJoueur(joueurCourant)
jusqu'a ce que coupGagnant(lePlateau,coup) ou totalementRempli(lePlateau)
si coupGagnant(lePlateau,coup) **alors**
etatPartie ← partieGagnee
quiAGagne ← autreJoueur(joueurCourant)
sinon
etatPartie ← partieNulle
finsi

fin

Conception détaillée 2 / 2

faireUnePartie - maintenant

procédure faireUnePartie **(E** afficher : AfficherPlateau , obtenirCoupJaune, obtenirCoupRouge : ObtenirCoup, **S** etatPartie : EtatPartie, quiAGagne : Pion**)**

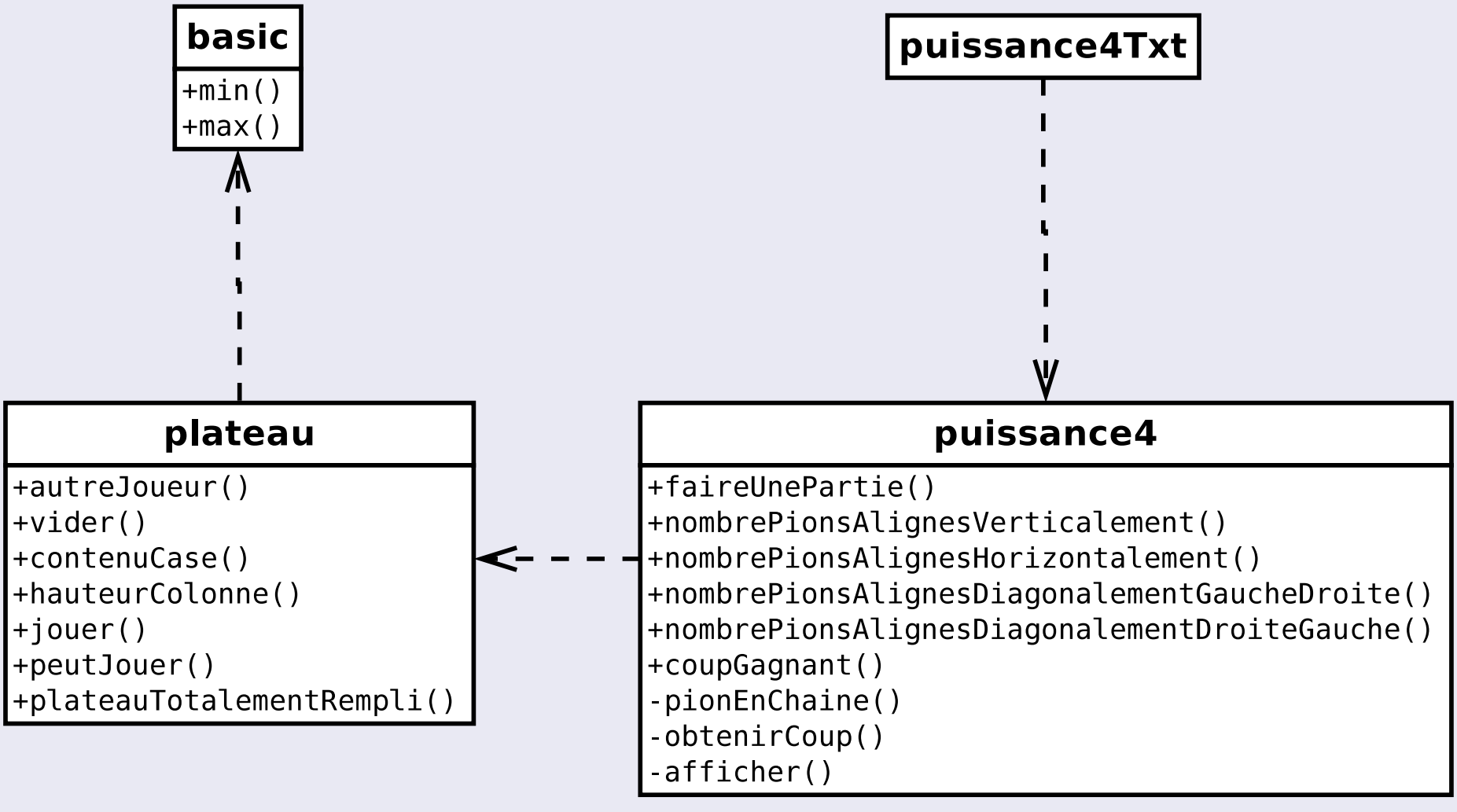
Déclaration lePlateau : Plateau, joueurCourant : Pion, coup : Colonne

```
debut
    vider(lePlateau)
    joueurCourant ← pionJaune
    afficher(lePlateau)
    repeter
        si joueurCourant=pionJaune alors
            coup ← obtenirCoupJaune(lePlateau,joueurCourant)
        sinon
            coup ← obtenirCoupRouge(lePlateau,joueurCourant)
        finsi
        jouer(lePlateau,coup,joueurCourant)
        afficher(lePlateau)
        joueurCourant ← autreJoueur(joueurCourant)
    jusqu'a ce que coupGagnant(lePlateau,coup) ou totalementRempli(lePlateau)
    si coupGagnant(lePlateau,coup) alors
        etatPartie ← partieGagnee
        quiAGagne ← autreJoueur(joueurCourant)
    sinon
        etatPartie ← partieNulle
    finsi
fin
```

Architecture des Systèmes d'Information

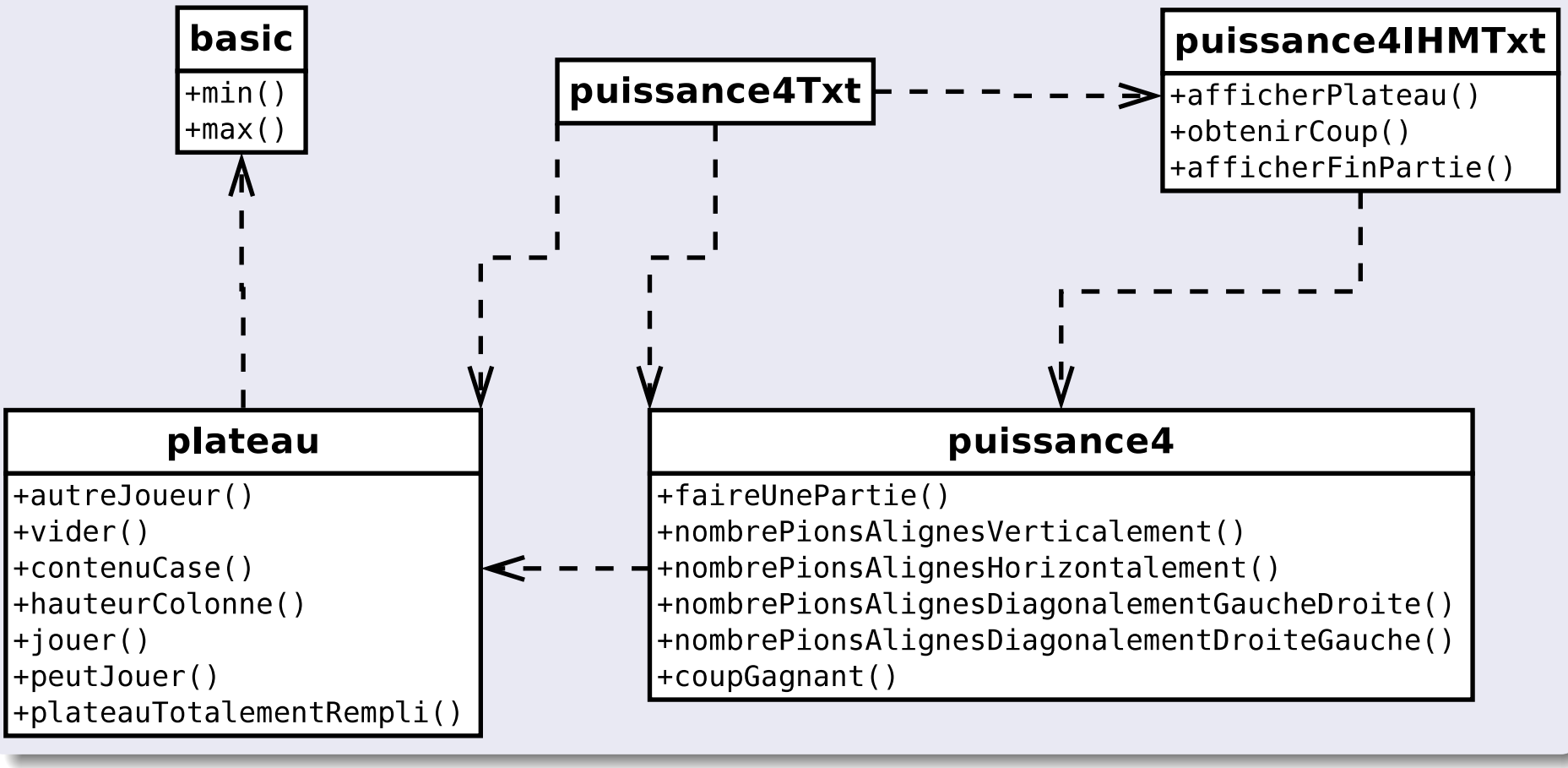
Développement 1 / 2

Diagramme d'unités - avant



Développement 2 / 2

Diagramme d'unités - maintenant



IHM Graphique

Questions liées au développement des interfaces graphiques modernes

- Comment positionner les éléments de l'IHM : positionnement absolu, positionnement relatif
- Comment capturer les multiples actions de l'utilisateur : boucle d'évènement, programmation par callback, paradigme de la programmation évènementiel
- Comment utiliser des ressources externes : interne au programme, externe au programme
- Comment afficher les composants graphiques de base : API de haut niveau et API de bas niveau

Architecture des Systèmes d'Information

SDL 1 / 2

Présentation

- *Simple DirectMedia Layer*
- « Bibliothèque *cross-platform* fournissant un accès bas niveau aux périphériques audio, d'entrée (clavier, souris, joystick) et graphiques via l'utilisation d'OpenGL et Direct3D. On l'utilise pour le développement de lecteur de vidéo, d'émulateur et de jeux vidéo »
- Disponible pour Windows, MacOS, Linux, iOS et Android
- Développé en C, mais utilisable directement en C++ et via des *binding* dans d'autres langages dont le Pascal
- Actuellement la version 2.0

SDL 2 / 2

Réponses aux questions

- Comment positionner les éléments de l'IHM : positionnement absolu
  ⇒ taille de fenêtre plutôt fixe, programmation ≪ lourde ≫
- Comment capturer les multiples actions de l'utilisateur : boucle d'évènements
  ⇒ s'adapte bien à la programmation des jeux et la séparation de la LM et de l'IHM par fonctions/procédures de callback métier
- Comment utiliser des ressources externes : externe au programme
  ⇒ devoir gérer des fichiers externes, ajout d'un répertoire ≪ ressources ≫
- Comment afficher les composants graphiques de base : API de bas niveau
  ⇒ programmation verbeuse, variables globales quasi obligatoires, utilisation intensive des pointeurs, ajout de fonctions/procédures pour avoir une programmation de plus haut niveau

SDL : principes de base 1 / 3

Type de base

- La fenêtre de l'application est de type `SDL_Window`. On obtient une fenêtre grâce à la fonction `SDL_CreateWindow`
- On ajoute des éléments graphiques (`SDL_Texture`) par l'intermédiaire d'un *render* (`SDl_Render`). On obtient le render d'une fenêtre grâce à la fonction `SDL_CreateRenderer`. Les objets ajoutés au *render* apparaissent à l'écran après l'appel à la procédure `SDL_RenderPresent`.
- Les textures sont créées à partir de surfaces (`SDL_Surface`) pour un *render* donné. On positionne un élément graphique sur un render grâce à la procédure `SDL_RenderCopy`.
- Les surfaces peuvent être créées à partir de divers sources : textes, images, etc. Une fois associé à un *render*, l'espace mémoire associé à une surface doit être libérée (procédure `SDL_FreeSurface`)

SDL : principes de base 2 / 3

Boucle d'évènements

- Boucle indéterministe : on sort de la boucle lorsque la partie est terminée
- On récupère un évènement grâce à la procédure `SDL_WaitEvent` et à la structure `SDL_Event`
- Les champs de la structure `SDL_Event` sont hiérarchisés.
  - On connaît la source de l'évènement grâce au champs `type` (exemple de valeur `SDL_QUIT`, `SDL_MOUSEBUTTONDOWN`, etc.)
  - Suivant la valeur du champs type, on utilise d'autres champs. Par exemple pour `SDL_MOUSEBUTTONDOWN`, on utilise le champ `button` de type `SDL_MouseButtonEvent` pour connaître les coordonnées de la souris, quel bouton a été appuyé, etc.

SDL : principes de base 3 / 3

Structure d'un programme SDL classique (simplifié)

1. Initialisation :
   - Création de la fenêtre : `SDL_CreateWindow`
   - Création du render : `SDL_CreateRenderer`
   - Création des textures : `SDL_CreateTextureFromSurface`
2. Boucle événementiel :
   - Prendre en compte les entrées
   - Mettre ajour l'affichage : effacer le render (`SDL_RenderClear`), mettre les textures (`SDL_RenderCopy`), demander l'affichage (`SDL_RenderPresent`)
3. Libération des espaces mémoire :
   - Libération des textures
   - Libération du *render*
   - Libération de la fenêtre
4. Quitter le programme

Binding Pascal

Caractéristiques

- *Binding* utilisé : Pascal-SDL-2[a] (`https://github.com/ev1313`), il fait « juste » le lien entre votre programme pascal et les fonctions C compilés dans `libSDL2.so` (ou `SDL2.dll`). Trois unités : `strings`, `sdl2` et `sdl2_ttf`
- Le nom des types des fonctions/procédures pascal sont les même qu'en C, excepté pour les types : précédés d'un T (par exemple `TSDL_Event`) ou d'un P (`PSDL_Event`)
- Les chaînes de caractères doivent être au format C (utilisation du type `PChar` et allocation dynamique)
- Les passages de paramètre en sortie (et quelque fois en entrée ! ? !) se font « à la C » (utilisation d'un pointeur, d'où utilisation de l'opérateur `@` plutôt que le passage par référence).

---

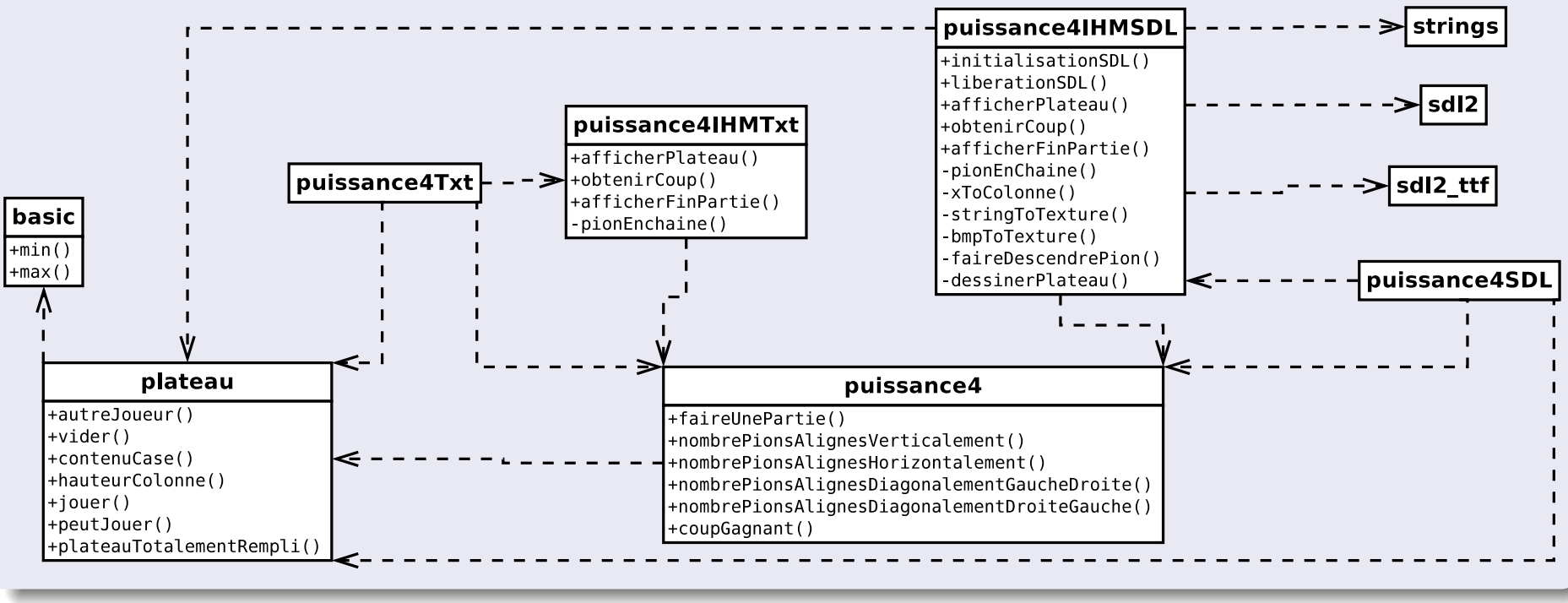
a. Ce n'est pas la *binding* officiel (il n'était pas disponible pour SDL2 au moment ou la version 1.0 de ce court a été rédigé)

IHM contrôlée par la LM

- Ce n'est pas le fonctionnement classique d'un programme SDL
- Création de boucles évènementielles lorsque la LM « donne la main » à l'IHM
- Organisation du programme :
  - Une unité `puissance4IHMSDL` qui utilise le binding SDL et qui propose 5 procédures/fonctions publiques :
    - deux d'initialisation et libération
    - trois équivalentes à celles vue dans l'unité `puissance4IHMTxt`
  - Un programme `puissance4SDL`, presque identique à `puissance4Txt`, qui utilise l'unité précédente

Architecture des Systèmes d'Information

Diagramme d'unités - Texte et graphique



puissance4IHMSDL.pas

Partie publique

```
1  unit puissance4IHMSDL;
2
3  interface
4
5  uses plateau,puissance4;
6
7  procedure initialisationSDL();
8  procedure liberationSDL();
9  procedure afficherPlateau(lePlateau :  TPlateau);
10 function obtenirCoup(lePlateau : TPlateau; joueur : TPion) : TColonne;
11 procedure afficherFinPartie(etatFinale : TEtatPartie; quiAGagne : TPion);
```

Partie privée

```
13 implementation
14
15 uses strings,sdl2,SDL2_ttf;
```

Architecture des Systèmes d'Information

puissance4IHMSDL.pas : Constantes

```
17 const
18    NOM_APPLICATION = 'Puissance 4';
19
20    LARGEUR_PLATEAU = 290;
21    HAUTEUR_PLATEAU = 290;
22
23    LARGEUR_PION = 30;
24    HAUTEUR_PION = 30;
25
26    HAUTEUR_FENETRE = HAUTEUR_PLATEAU+20;
27    LARGEUR_FENETRE = LARGEUR_PLATEAU;
28
29    BORD = 10;
30
31    PAS_DESCENTE = 4;
32    TEMPS_ATTENTE_MESSAGE_FINAL = 2000;
33
34    REPERTOIRE_IMAGES = 'ressources/img/';
35    REPERTOIRE_FONTS = 'ressources/fonts/';
36
37    FICHIER_FONT_INTERACTION = REPERTOIRE_FONTS+'Arial.ttf';
38    TAILLE_FONT_INTERACTION = 12;
39
40    FICHIER_FONT_MESSAGE_FINAL = REPERTOIRE_FONTS+'Comic_Sans_MS.ttf';
41    TAILLE_FONT_MESSAGE_FINAL = 25;
42
43    FICHIER_IMAGE_PLATEAU = REPERTOIRE_IMAGES+'jeu.bmp';
44    FICHIER_IMAGE_PION_JAUNE = REPERTOIRE_IMAGES+'pionJaune.bmp';
45    FICHIER_IMAGE_PION_ROUGE = REPERTOIRE_IMAGES+'pionRouge.bmp';
```

puissance4IHMSDL.pas : Variables globales

```
47 var
48    fenetrePrincipale : PSDL_Window;
49    renderer : PSDL_Renderer;
50
51    texPionRouge : PSDL_Texture;
52    texPionJaune : PSDL_Texture;
53    texPlateau: PSDL_Texture;
54
55    fontInteraction, fontMessageFinal : PTTF_Font;
```

Architecture des Systèmes d'Information

puissance4IHMSDL.pas : abstraction 1 / 2

```
71 function stringToPchar(s : String): PChar;
72 begin
73    stringToPchar := StrAlloc(length(s)+1);
74    StrPCopy(stringToPchar,s);
75 end;
76
77 function bmpToTexture(filename : String; renderer : PSDL_Renderer; transparence :
       Boolean; rTransparence, gTrasnparence, bTransparence : UInt8):PSDL_Texture;
78 var
79    bmp : PSDL_Surface;
80    pfichier : PChar;
81    couleur : UInt32;
82 begin
83    pfichier := stringToPchar(filename);
84    bmp := SDL_LoadBMP(pfichier);
85    couleur := SDL_MapRGB(bmp^.format, rTransparence, gTrasnparence, bTransparence);
86    if transparence then
87       SDL_SetColorKey(bmp, 1, couleur);
88    bmpToTexture := SDL_CreateTextureFromSurface(renderer, bmp);
89    SDL_FreeSurface(bmp);
90    StrDispose(pfichier)
91 end;
```

Architecture des Systèmes d'Information

puissance4IHMSDL.pas : abstraction 2 / 2

```
 93 procedure stringToTexture(s : String; font : PTTF_Font; r,g,b : UInt8; var resultat :
        PSDL_Texture; var h,w : SInt32);
 94 var
 95    texte : PChar;
 96    surf : PSDL_Surface;
 97    couleur : TSDL_Color;
 98 begin
 99    texte := stringToPchar(s);
100    couleur.r := r;
101    couleur.g := g;
102    couleur.b := b;
103    couleur.unused := 255;
104    surf := TTF_RenderText_Solid(font, texte, couleur);
105    h := surf^.h;
106    w := surf^.w;
107    resultat := SDL_CreateTextureFromSurface(renderer, surf);
108    SDL_FreeSurface(surf);
109    StrDispose(texte)
110 end;
```

puissance4IHMSDL.pas : affichage/animation 1 / 3

```
113 procedure dessinerPlateau(lePlateau : TPlateau);
114 var
115    i : TColonne;
116    j : TLigne;
117    posPion : TSDL_Rect;
118    texPion : PSDL_Texture;
119    posPlateau : TSDL_Rect = (x:0;y:0;w:HAUTEUR_PLATEAU;h:LARGEUR_PLATEAU);
120 begin
121    posPion.w := HAUTEUR_PION;
122    posPion.h := LARGEUR_PION;
123    SDL_RenderCopy(renderer, texPlateau, Nil, @posPlateau);
124    for j:=1 to NB_LIGNES do
125       for i:=1 to NB_COLONNES do
126       begin
127          posPion.x := BORD + (i-1) * (LARGEUR_PION + BORD);
128          posPion.y := HAUTEUR_PLATEAU - (j * (LARGEUR_PION + BORD));
129          if contenuCase(lePlateau,i,j) <> vide then
130          begin
131             if contenuCase(lePlateau,i,j) = pionJaune then
132                texPion := texPionJaune
133             else
134                texPion := texPionRouge;
135             SDL_RenderCopy(renderer, texPion, Nil, @posPion);
136          end
137       end
138 end;
```

puissance4IHMSDL.pas : affichage/animation 2 / 3

```
140 procedure faireDescendrePion(lePlateau : TPlateau; colonne : TColonne; pion : TPion);
141 var
142    posPion : TSDL_Rect;
143    posPlateau : TSDL_Rect = (x:0; y:0; w: HAUTEUR_PLATEAU; h: LARGEUR_PLATEAU);
144    i : Sint32;
145    texPion : PSDL_Texture;
146 begin
147    if pion = pionJaune then
148       texPion := texPionJaune
149    else
150       texPion := texPionRouge;
151    posPion.x := BORD + (LARGEUR_PION + BORD) * (colonne -1);
152    posPion.w := HAUTEUR_PION;
153    posPion.h := LARGEUR_PION;
154    for i:=0 to (HAUTEUR_PLATEAU - HAUTEUR_PION * (hauteurColonne(lePlateau,colonne) +
          2)) div PAS_DESCENTE do
155    begin
156       SDL_RenderClear(renderer);
157       dessinerPlateau(lePlateau);
158       posPion.y := PAS_DESCENTE*i;
159       SDL_RenderCopy(renderer, texPion, Nil, @posPion);
160       SDL_RenderCopy(renderer, texPlateau, Nil, @posPlateau);
161       SDL_RenderPresent(renderer)
162    end
163 end;
```

Architecture des Systèmes d'Information

puissance4IHMSDL.pas : affichage/animation 3 / 3

```
165 procedure afficherFinPartie(etatFinale : TEtatPartie; quiAGagne : TPion);
166 var
167    message : String;
168    texMessage : PSDL_Texture;
169    pos : TSDL_Rect;
170 begin
171    if etatFinale=partie_gagnee then
172       message := 'Les␣'+pionEnChaine(quiAGagne)+'␣ont␣gagne'
173    else
174       message := 'Partie␣nulle';
175    stringToTexture(message,fontMessageFinal,70,200,70,texMessage,pos.h,pos.w);
176    pos.y := (HAUTEUR_FENETRE - pos.h) div 2;
177    pos.x := (LARGEUR_FENETRE - pos.w) div 2;
178    SDL_RenderCopy(renderer,texMessage,Nil,@pos);
179    SDL_RenderPresent(renderer);
180    sdl_delay(TEMPS_ATTENTE_MESSAGE_FINAL)
181 end;
```

puissance4IHMSDL.pas : *callback* 1 / 2

```
184 procedure afficherPlateau(lePlateau :  TPlateau);
185 begin
186    SDL_RenderClear(renderer);
187    dessinerPlateau(lePlateau);
188    SDL_RenderPresent(renderer)
189 end;


191 function obtenirCoup(lePlateau : TPlateau; joueur : TPion) : TColonne;
192 var
193    OK     : Boolean;
194    choix : Integer;
195    terminer : Boolean;
196    evenements: TSDL_Event;
197    posTexte : TSDL_Rect = (x:0;y:HAUTEUR_PLATEAU+2;w:0;h:0);
198    texTexte : PSDL_Texture;
199 begin
200    stringToTexture('Au joueur qui a les '+pionEnChaine(joueur)+' de jouer.',
         fontInteraction,0,0,0,texTexte,posTexte.h,posTexte.w);
201    repeat
202       SDL_RenderClear(renderer);
203       dessinerPlateau(lePlateau);
204       SDL_RenderCopy(renderer,texTexte,Nil,@posTexte);
205       SDL_RenderPresent(renderer);
206       terminer := false;
```

Architecture des Systèmes d'Information

puissance4IHMSDL.pas : *callback* 2 / 2

```
207       while not terminer do
208       begin
209          SDL_WaitEvent(@evenements);
210          if evenements.type_ = SDL_QUITEV  then
211          begin
212             liberationSDL();
213             halt()
214          end;
215          if evenements.button.type_ = SDL_MOUSEBUTTONDOWN then
216          begin
217             choix := xToColonne(evenements.motion.x);
218             terminer := true;
219          end
220       end;
221       if (choix>=1) and (choix<=Plateau.NB_COLONNES) then
222       begin
223          obtenirCoup:=choix;
224          OK:=Plateau.peutJouer(lePlateau,obtenirCoup)
225       end else
226          OK:=False;
227    until OK;
228    faireDescendrePion(lePlateau,obtenirCoup,joueur)
229 end;
```

puissance4IHMSDL.pas : initialisation et libération SDL 1 / 2

```
232 procedure initialisationSDL();
233 var
234    nom_font : PChar;
235 begin
236    SDL_INIT(SDL_INIT_VIDEO);
237    TTF_Init();
238
239    nom_font := stringToPchar(FICHIER_FONT_INTERACTION);
240    fontInteraction := TTF_OpenFont(nom_font,TAILLE_FONT_INTERACTION);
241    StrDispose(nom_font);
242
243    nom_font := stringToPchar(FICHIER_FONT_MESSAGE_FINAL);
244    fontMessageFinal := TTF_OpenFont(nom_font,TAILLE_FONT_MESSAGE_FINAL);
245    StrDispose(nom_font);
246
247    fenetrePrincipale := SDL_CreateWindow(NOM_APPLICATION, 100, 100, LARGEUR_FENETRE,
         HAUTEUR_FENETRE, SDL_WINDOW_SHOWN);
248
249    renderer := SDL_CreateRenderer(fenetrePrincipale, -1,
250                                SDL_RENDERER_ACCELERATED or SDL_RENDERER_PRESENTVSYNC);
251    SDl_SetRenderDrawColor(renderer,255,255,255,0);
252
253    texPionRouge := bmpToTexture(FICHIER_IMAGE_PION_ROUGE,renderer, false,0,0,0);
254    texPionJaune := bmpToTexture(FICHIER_IMAGE_PION_JAUNE,renderer, false,0,0,0);
255    texPlateau := bmpToTexture(FICHIER_IMAGE_PLATEAU,renderer,true,255,255,255)
256 end;
```

puissance4IHMSDL.pas : initialisation et libération SDL 2 / 2

```
258 procedure liberationSDL();
259 begin
260     TTF_CloseFont(fontInteraction);
261     TTF_CloseFont(fontMessageFinal);
262     TTF_Quit();
263     SDL_DestroyRenderer(renderer);
264     SDL_DestroyWindow(fenetrePrincipale);
265     SDL_QUIT;
266 end;
```

puissance4SDL.pas

```
1  program puissance4SDL;
2
3  uses plateau,puissance4,puissance4IHMSDL;
4
5  var
6     etatFinale : TEtatPartie;
7     quiAGagne  : TPion;
8  begin
9     initialisationSDL();
10    Puissance4.faireUnePartie(@obtenirCoup,@obtenirCoup,@afficherPlateau,etatFinale,
          quiAGagne);
11    afficherFinPartie(etatFinale,quiAGagne);
12    liberationSDL()
13 end.
```

Architecture des Systèmes d'Information

Conclusion

Critères d'un bon programme

lisible	✓
fiable	✓
maintenable	✓
réutilisable	✓
portable	✓
correct (preuve)	✗
efficace (complexité)	✓
faire face à des contraintes "économiques"	✓

asi
Architecture des Systèmes d'Information