Modalités du CC2

- L'étudiant dispose du temps d'autonomie supervisé en classe par l'enseignant (pendant lequel celui-ci se tient disponible pour reformuler une incompréhension, donner des pistes de résolution en cas de blocage, guider l'étudiant à la résolution d'éventuelles problématiques).

- S'il le souhaite, l'étudiant a la possibilité d'avancer individuellement et sans assistance sa production sur le temps non dédié au cours.

- Il est conseillé de maîtriser les exercices d'entraînement (à rendre) jusqu'à la section 5 (boucles) pour réussir les exercices proposés.

- Chaque exercice doit être représenté par un code source rendu dans une archive `.zip` sous forme de fichiers sources en **Langage C** en extension `.c` (qui compilent). Soyez rigoureux sur votre rendu.

- Le code à produire par l'étudiant pour réussir les exercices s'attend à la restitution des notions étudiées jusqu'à la section 5 (un code plus avancé est fourni : exemple du mini-projet).

- L'évaluation vérifiera et valorisera des éléments de code, d'efficacité et d'adéquation de la solution proposée (entrées, sorties, logique conditionnelle, validité d'opérations, inclusions et autres éléments pertinents liés au langage C et à la logique du code proposé).

- L'utilisation d'outils ou moyens se substituant à la connaissance ou aux compétences de l'étudiant sera pénalisée en cas de détection par l'enseignant (niveau de l'étudiant dissimulé et donc non évaluable : 0).

- Le rendu d'un travail par groupe est autorisé uniquement selon les modalités exprimées dans le support de cours (perte de 4 points par participant supplémentaire, note du CC2 lourdement pénalisée en cas doute sur un rendu potentiellement malhonnête ou non déclaré).

- Le devoir est à rendre sur **MyGES** au plus tard le **Dimanche 04 Février** 2024 à 23h59.

Bon courage !

Assiduité (4 points)

Votre rendu doit contenir vos traces de recherche / solutions pour les exercices d'entraînement suivants :

- Section 4 (conditions) : exercices 18 à 25 (2 points).
- Section 5 (boucles) : exercices 26 à 30 (2 points).

Attendus :

- Traces de recherche et de compréhension.
- Proposition d'une solution personnelle valide.

Compréhension (4 points)

Alice a découvert l'instruction `goto`. Elle a demandé à une intelligence artificielle de produire un code l'utilisant et se demande pourquoi ceci n'est pas enseigné dans un cours pour débutant. Alice ne sait maintenant plus ce que fait son code et a des difficultés à le relire :

```c
#include <stdio.h>
#include <stdlib.h>

int main() {
  int i;
  int j;
  int k;
  printf("Entrez une valeur : ");
  goto i;
  f1s : i = 0;
  f1c : if(i >= k)
  goto f1e; ++i;
  f2s : j = 0;
  f2c : if(j >= i)
  goto f2e; ++j;
  printf("*");
  goto f2c;
  i : scanf("%d", &k);
  goto f1s;
  f2e : printf("\n");
  goto f1c;
  f1e : exit(EXIT_SUCCESS);
}
```

Aider Alice :

- Expliquer ce que fait son code.
- Produire une version lisible et sans `goto` de son code.
- Expliquer ce qui fait en général du `goto` une mauvaise pratique si non utilisé avec parcimonie.
- Proposer des cas d'usage où le `goto` pourrait être utilisé intelligemment.

Attendus :

- Comprendre un code incluant un nouveau concept.
- Réécrire un code fonctionnel dans une version lisible.
- Comprendre les limitations d'un concept.
- Comprendre l'intérêt d'un concept.

Implémentation (4 points)

Bob a découvert l'étrange inscription suivante sur un tableau dans une salle de classe :

$$(IBE)_x = (C4)_y$$
$$(27D84B2G)_x\,(6A0HDJ0BC2)_x$$

Il a le souvenir d'avoir vu les bases binaires, octales, décimales et hexadécimales. Il aimerait que vous lui fabriquiez un programme pour convertir une valeur d'une base à une autre peu importe les bases entières fournies.

Une sortie pourrait ressembler à la suivante :

```
Entrez la base d'expression : 22
Entrez la base attendue : 123
Valeur en base 22 : c4
Valeur en base 123 : 2m
```

Attendus :

- Lire une valeur dans une base entière quelconque.
- Écrire une valeur dans une base entière quelconque.

Performances (4 points)

Charlie aimerait créer une intelligence artificielle pour battre Oscar dans un jeu. Il a réussi à déterminer la fonction qui règle la hauteur du terrain dans le jeu d'Oscar. Tous les moments où cette fonction passe par 0, le personnage rencontrerait de l'eau : ceci change son fonctionnement.

Déterminer tous les moments où la fonction suivante passe par 0 pour les valeurs de x entre 0 et 10 000 :

$$x \mapsto \frac{1}{2}\left(200\sin\left(\frac{x}{400}\right) + 100\sin\left(\frac{x}{117} + 25\sin\left(\frac{x}{51}\right)\right) + 10\sin\left(\frac{x}{12}\right) + 10\sin\left(\frac{x+10}{13}\right)\right)$$

Attendus :

- Votre méthode calcule les 19 solutions au problème (2 points : proportionnel au nombre de solutions trouvées).
- Votre méthode calcule les solutions en moins de 50 millisecondes, tranches de points :
  - moins de 50 ms : 2 points
  - moins de 200 ms : 1.5 points
  - moins de 1 s : 1 point
  - moins de 10 s : 0.5 point

Mini-projet (4 points)

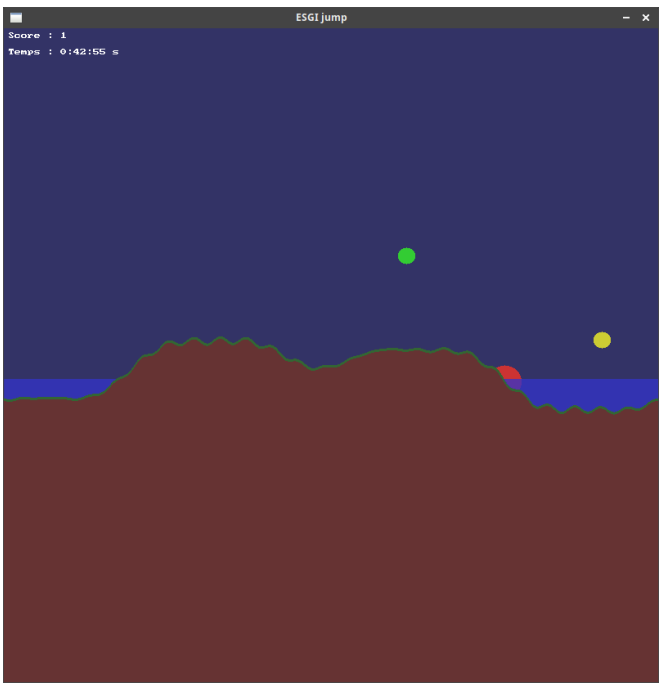
Oscar souhaite créer un jeu où le joueur pourrait se déplacer sur un terrain vu de côté. A priori, Charlie aurait trouvé la formule qui souhaite utiliser pour modéliser le terrain.

Dans son jeu, le joueur incarne une boule verte, doit collecter une boule jaune et éviter une boule rouge. Le joueur se dirige au clavier et peut sauter. S'il se trouve dans l'eau, la physique change. Oscar propose d'utiliser un modèle physique simple pour gérer le saut et la nage : basé sur position, vitesse et accélération.

$$\texttt{vitesse}_{i+1} \leftarrow \texttt{vitesse}_i + \texttt{accélération}_{i+1}$$

$$\texttt{position}_{i+1} \leftarrow \texttt{position}_i + \texttt{vitesse}_{i+1}$$

Il aimerait que le jeu ressemble au moins à l'image suivante :



Le joueur serait déplacé sur les côtés en jouant sur la vitesse horizontale et une imagination de la gravité (saut) / poussée d'Archimède (eau) sur l'accélération verticale.
Il vous a fait le code graphique et aimerait que vous complétiez les `TODO` avec des instructions (voir l'initialisation de SDL, section 15 dans le cours).

```c
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <time.h>
#include <SDL/SDL.h>
#include <SDL/SDL_gfxPrimitives.h>

/* Paramètres de la fenêtre : */
const int largeur = 800;
const int hauteur = 800;
const char * titre = "ESGI jump";

/* Coordonnées du joueur (position) : */
float px, py;
/* Vitesse du joueur (déplacement) : */
float vx, vy;
/* Accélération du joueur (saut, eau) : */
float ax, ay;

/* Coordonnées de l'objectif : */
float ox, oy;
/* Coordonnées de l'adversaire : */
float ex, ey;

/* Position de la caméra et échelle de la vue : */
float cx, cy, cz;

float distance(float first_x, float first_y, float second_x, float
↪  second_y) {
  float d = 0;
  /* TODO : calculer la distance entre deux position. */
  return d;
}

float hauteur_terrain(float x) {
  float h = 500 - x;
  /* TODO : proposer une modélisation du terrain. */
```

```
  return h;
}

void placer_objectif() {
  /* TODO : Placer l'objectif aléatoirement. */
  ox = 0;
  oy = 0;
}

void placer_adversaire() {
  /* TODO : Placer l'adversaire aléatoirement. */
  ex = 0;
  ey = 0;
}

int ecran_depuis_camera_x(float x) {
  /* TODO : Déterminer l'abscisse sur l'écran depuis une position
  ↪  sur le terrain. */
  return x + 100;
}

int ecran_depuis_camera_y(float y) {
  /* TODO : Déterminer l'ordonnée sur l'écran depuis une position
  ↪  sur le terrain. */
  return y + 100;
}

float camera_depuis_ecran_x(int x) {
  /* TODO : Déterminer l'abscisse sur le terrain depuis une position
  ↪  sur l'écran */
  return x - 100;
}

float camera_depuis_ecran_y(int y) {
  /* TODO : Déterminer l'ordonnée sur le terrain depuis une position
  ↪  sur l'écran */
  return y - 100;
}

float ecran_depuis_camera_z(int z) {
  /* TODO : Déterminer la mesure sur l'écran depuis sa mesure sur le
  ↪  terrain. */
  return z;
```

```
}

float camera_depuis_ecran_z(int z) {
  /* TODO : Déterminer la mesure sur le terrain depuis sa mesure sur
  ↪  l'écran. */
  return z;
}

void affichage(SDL_Surface * ecran) {
  /* Remplissage de l'écran par un gris foncé uniforme : */
  SDL_FillRect(ecran, NULL, SDL_MapRGB(ecran->format, 51, 51, 102));
  /* Affichage du joueur : */
  filledCircleRGBA(ecran, ecran_depuis_camera_x(px),
  ↪  ecran_depuis_camera_y(py), ecran_depuis_camera_z(25), 51, 204,
  ↪  51, 255);

  filledCircleRGBA(ecran, ecran_depuis_camera_x(ox),
  ↪  ecran_depuis_camera_y(oy), ecran_depuis_camera_z(25), 204,
  ↪  204, 51, 255);

  filledCircleRGBA(ecran, ecran_depuis_camera_x(ex),
  ↪  ecran_depuis_camera_y(ey), ecran_depuis_camera_z(50), 204, 51,
  ↪  51, 255);

  int x, y, z;
  for(x = 0; x < largeur; ++x) {
    y =
    ↪  ecran_depuis_camera_y(hauteur_terrain(camera_depuis_ecran_x(x)));
    z = ecran_depuis_camera_y(0);
    boxRGBA(ecran, x, y, x + 1, z, 51, 51, 204, 127);
    boxRGBA(ecran, x, y, x + 1, hauteur, 102, 51, 51, 255);
    boxRGBA(ecran, x, y, x + 1, y + ecran_depuis_camera_z(5), 51,
    ↪  102, 51, 255);
  }
}

int calculs(int score) {
  /* TODO : déplacer le joueur. */
  /* TODO : ajouter 1 au score si le joueur atteint l'objectif et le
  ↪  replacer. */
  /* TODO : retirer 10 au score si le joueur est touché par
  ↪  l'adversaire et le replacer. */
  /* TODO : déplacer l'adversaire. */
```

```c
  return score;
}

void action_jump() {
  /* TODO : enclencher un saut. */
}

void action_droite() {
  /* TODO : déplacer le joueur vers la droite. */
}

void action_gauche() {
  /* TODO : déplacer le joueur vers la gauche. */
}

void action_sans_direction() {
  /* TODO : ne plus se diriger vers une direction. */
}

void deplacer_camera() {
  /* TODO : si le joueur sort ou est proche de sortir du champ de
  ↪  vision, le suivre. */
}

int main() {
  srand(time(NULL));
  /* Création d'une fenêtre SDL : */
  if(SDL_Init(SDL_INIT_EVERYTHING) != 0) {
    fprintf(stderr, "Error in SDL_Init : %s\n", SDL_GetError());
    exit(EXIT_FAILURE);
  }
  SDL_Surface * ecran = NULL;
  if((ecran = SDL_SetVideoMode(largeur, hauteur, 32, SDL_HWSURFACE |
  ↪  SDL_DOUBLEBUF)) == NULL) {
    fprintf(stderr, "Error in SDL_SetVideoMode : %s\n",
    ↪  SDL_GetError());
    SDL_Quit();
    exit(EXIT_FAILURE);
  }
  SDL_WM_SetCaption(titre, NULL);

  /* Placement du joueur au centre de la fenêtre : */
  px = 0;
```

```
py = 0;
vx = 0;
vy = 0;
ax = 0;
ay = 0;

cx = 0;
cy = 0;
cz = 500;

int active = 1;
SDL_Event event;
int moving_right = 0;
int moving_left = 0;
int jump = 0;

int score = 0;
unsigned int temps;
char display[100];

placer_objectif();

while(active) {

  temps = SDL_GetTicks();
  affichage(ecran);
  sprintf(display, "Score : %d", score);
  stringRGBA(ecran, 5, 5, display, 255, 255, 255, 255);
  sprintf(display, "Temps : %u:%02u:%02u s", (temps / 1000) / 60,
  ↪  (temps / 1000) % 60, (temps % 1000) / 10);
  stringRGBA(ecran, 5, 25, display, 255, 255, 255, 255);
  SDL_Flip(ecran);

  while(SDL_PollEvent(&event)) {

    switch(event.type) {
      /* Utilisateur clique sur la croix de la fenêtre : */
      case SDL_QUIT : {
        active = 0;
      } break;

      /* Utilisateur enfonce une touche du clavier : */
      case SDL_KEYDOWN : {
```

```
          switch(event.key.keysym.sym) {
            /* Touche Echap : */
            case SDLK_ESCAPE : {
              active = 0;
            } break;

            case SDLK_z :
            case SDLK_UP : {
              if(! jump) {
                action_jump();
              }
              jump = 1;
            } break;

            case SDLK_d :
            case SDLK_RIGHT : {
              moving_right = 1;
            } break;

            case SDLK_q :
            case SDLK_LEFT : {
              moving_left = 1;
            } break;
          }
        } break;

        case SDL_KEYUP : {
          switch(event.key.keysym.sym) {
            case SDLK_z :
            case SDLK_UP : {
              jump = 0;
            } break;

            case SDLK_d :
            case SDLK_RIGHT : {
              moving_right = 0;
            } break;

            case SDLK_q :
            case SDLK_LEFT : {
              moving_left = 0;
            } break;
          }
```

```
        } break;

        case SDL_MOUSEBUTTONDOWN : {
          switch(event.button.button) {
            case SDL_BUTTON_WHEELUP : {
              cz *= 0.8;
              if(cz < 1) {
                cz = 1;
              }
            } break;

            case SDL_BUTTON_WHEELDOWN : {
              cz /= 0.8;
              if(cz > 10000) {
                cz = 10000;
              }
            } break;
          }
        } break;
      }
    }

    if(moving_right && moving_left) {
      action_sans_direction();
    } else if(moving_right) {
      action_droite();
    } else if(moving_left) {
      action_gauche();
    } else {
      action_sans_direction();
    }
    score = calculs(score);

    deplacer_camera();

    if(score >= 10) {
      affichage(ecran);
      sprintf(display, "Score : %d", score);
      stringRGBA(ecran, 5, 5, display, 255, 255, 255, 255);
      sprintf(display, "Temps : %u:%02u:%02u s", (temps / 1000) /
      ↪  60, (temps / 1000) % 60, (temps % 1000) / 10);
      stringRGBA(ecran, 5, 25, display, 255, 255, 255, 255);
      stringRGBA(ecran, largeur / 2 - 10, hauteur / 2, "BRAVO !",
      ↪  255, 255, 255, 255);
```

```
      SDL_Flip(ecran);
      SDL_Delay(3000);
      active = 0;
    }

    SDL_Delay(1000 / 60);
  }

  SDL_FreeSurface(ecran);
  SDL_Quit();
  exit(EXIT_SUCCESS);
}
```

Attendus :

- Déplacement du joueur au clavier.
- Déplacement de l'adversaire vers le joueur.
- Interactions entre joueur et objectif.
- Interactions entre joueur et adversaire.
- Effet saut du joueur.
- Effet poussée d'Archimède et ralentissement dans l'eau.