Introduction to Programming

—

Final examination on machine

—

G1: clement.riu(at)enpc.fr
G2: marie.haghebaert(at)inrae.fr
G3: thomas.belos(at)enpc.fr
G4: abderahmane.bedouhene(at)enpc.fr
G5: youval.vanlaer(at)enpc.fr
G6: pascal.monasse(at)enpc.fr
G7: thomas.daumain(at)enpc.fr

—

14/01/22

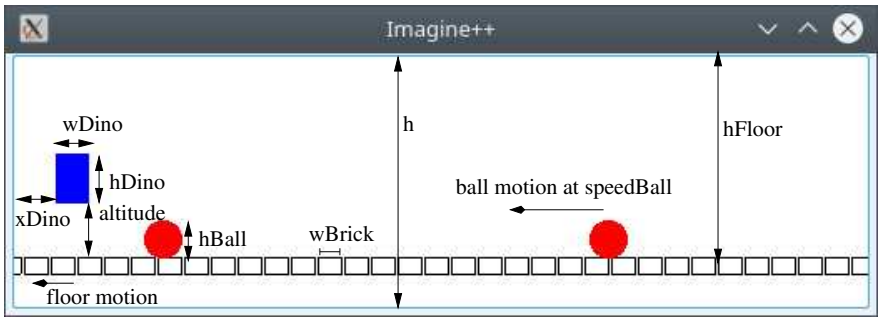
—

1 Instructions

1.1 Dinosaur Game

The goal is to build a simplified version of the Dinosaur Game, a basic game included in the Google Chrome web browser. A dinosaur has to avoid obstacles moving its way by jumping. Our dinosaur will be represented by a rectangle and the obstacles are balls. The game is registered so that it can be viewed backward and forward after the user lost.
Important: All methods of classes take a time `t0` (integer) as parameter.

> **It is more important to deliver a clean code (commented and correctly indented) that *compiles* than answering all questions. For that, check after each step that the build works. At the end, create an archive with source code and file `CMakeLists.txt` to upload on educnet.**




```
int keyboard() {
  Event e;
  do {
    getEvent(0,e);
    if(e.type == EVT_KEY_ON)
      return e.key;
  } while(e.type!=EVT_NONE);
  return 0;
}
```

Figure 1: Blue jumping dinosaur and red balls. Altitude is variable (when the dinosaur jumps), computed question 4. The function `keyboard` returns the pressed key code (without waiting), 0 if none. You can copy-paste it from Practical#8.

1.2 Happy jumping dino

1. Create a new project and a basic `main` function opening a window. In a separate file `dino.h` write the constants: wDino=20 and hDino=30 the dimensions of the dinosaur, xDino=25 the abscissa of the dinosaur (fixed, the decor is moving), $w \times h$=512 × 5∗hDino the window dimensions, hFloor=h-hDino the base ordinate of the dinosaur when not jumping.

2. Write a class `Dino` and its constructor. It only needs an integer `t` storing the time of the last jump start. A jump lasts for a constant time tJump=20 and reaches a height hJump=3*hDino (yes, the dinosaur is heavy but is a high jumper!). Initially, the dinosaur is not jumping, so we put its `t` as sufficiently negative.

3. Method `jump` registers that a jump starts. Method `jumping` indicates if the jump is still in process.

4. Method `altitude` computes the altitude above the floor (0 if not jumping). According to Newton's gravitation law, it has a parabolic evolution given by equation

$$h = \text{hJump} * \left(1 - \left(1 - 2\frac{t_0 - t}{\text{tJump}}\right)^2\right). \tag{1}$$

   (Proof: $t_0 = t \Rightarrow h = 0$, $t_0 = t + \text{tJump}/2 \Rightarrow h = \text{hJump}$, $t_0 = t + \text{tJump} \Rightarrow h = 0$)

5. In a separate file, define a class `Recorder`. It has a field of type `Dino`. First this class will be used to play the game, the recording part will be coded in Section 1.4.

6. Method `Recorder::display` clears the window and draws the dinosaur at the current time.

7. Method `Recorder::action` calls the function `keyboard`: if the space bar key is pressed, the dinosaur is set to start jumping but only if the last jump has finished.

8. In the `main`, let the user make the dino jump on demand.

9. `Recorder::display` draws the floor, composed of disjoint rectangles of width wBrick=16 and height 10 pixels. The floor is shown moving to the left (as dinosaur moves to the right), the bricks are shifted 1 pixel at each time increment.

1.3 Life becomes harder with moving balls

10. A ball has diameter hBall=3/4 hDino and will be moving with a constant speed speedBall=8 pixels per time increment. It stores a time and abscissa for an initial position. Write class `Ball` in `dino.h` and a constructor, initially at abscissa 1000 for time 0.

11. `Ball::set` records a time and abscissa as initial position.

12. `Ball::center` returns the abscissa of its center at current time. It is based on initial position and speed, moving to the left (toward the poor dino).

13. `Ball::reInit` is used to recycle a ball that was dodged by the jumping dino if it went out of screen: it restarts at abscissa xBase (a method parameter) plus a random gap between one and three times tJump*speedBall pixels, but it must appear to the right, so that the result is set to at least `w` (the window width). The function returns true if recycling took place.

14. Add an array of nBalls=3 balls (nBalls is a constant) in class `Recorder`. In the constructor of the class, the ball are regularly spaced from abscissa w with a space of tJump*speedBall pixels.

15. Insert the display of balls in `Recorder::display` and call `reInit` on all balls in method `action`. The parameter xBase of `reInit` is the position of preceding ball.

16. In the main, let the user play the game with a span of 20 milliseconds for each time increment.

17. Add a method `Dino::crash` taking a ball and indicating whether the ball intersects the rectangle of the dinosaur. The squared distance to the ball center can be computed

$$d^2 = \max(0, h - \text{hBall}/2)^2 + \max(0, \text{xDino} - c)^2 + \max(0, c - \text{xDino} - \text{wDino})^2, \tag{2}$$

   with h the altitude of the dino and c the abscissa of the ball center.

18. Add method `Recorder::crash` indicating if the dino crashes with one of the balls. Insert the crash test in the game, letting it finish when it happens.

1.4 Recorder replay

19. Add a structure `Action`, a triplet (t, x, i) with t the time, x the abscissa and i the ball number. An action is either a dino jump start (then x and i are set negative) or a ball reinitialization.

20. The recorder will store all actions during the game. Insert a dynamic array `actions` in `Recorder` reserving initially the space for one action. Modify constructor and destructor accordingly. The management of the array follows this principle: `nmax` actions are initially allocated, but $n = 0$, the actual number of stored actions.

21. Write method `Recorder::record` storing a new action: if n=nmax, double nmax and reallocate the array to leave space for the new action.

22. In `Recorder::action`, store actions if they happen (jump and reinitialization of a ball). In the constructor, record also actions for the initial positions of the balls.

23. Write `Recorder::set` taking a time and resetting the game at this time: find the preceding stored jump in recorded actions, and for each ball the preceding reinitialization.

24. When the game is finished, let the user visualize it back and forth by arrow keys.