Snake Game

9&10 - Danh sách liên kết
https://github.com/tqlong/advprogram

Nội dung

- **Trò chơi: Snake**
- Sân chơi
  - Mảng 2 chiều
- Con rắn
  - Danh sách liên kết có đuôi
- Bắt phím di chuyển rắn
  - SDL_PollEvent()
- Xử lý va chạm

Trò chơi Snake

- Sân chơi hình chữ nhật
  - Trên sân chơi xuất hiện các quả cherry ngẫu nhiên
- Rắn lúc đầu
  - dài 01 ô (tính cả đầu), ở giữa màn hình, đi xuống
- Người chơi điều khiển rắn di chuyển bằng các phím mũi tên
- Mỗi lần rắn ăn 1 quả cherry thì dài thêm 1 ô
  - Thử sức: nhiều loại quả, mỗi loại một tác dụng
- Rắn va phải tường hoặc chính nó → thua
  - https://www.youtube.com/watch?v=kTIPpbIbkos

Các tác vụ của trò chơi

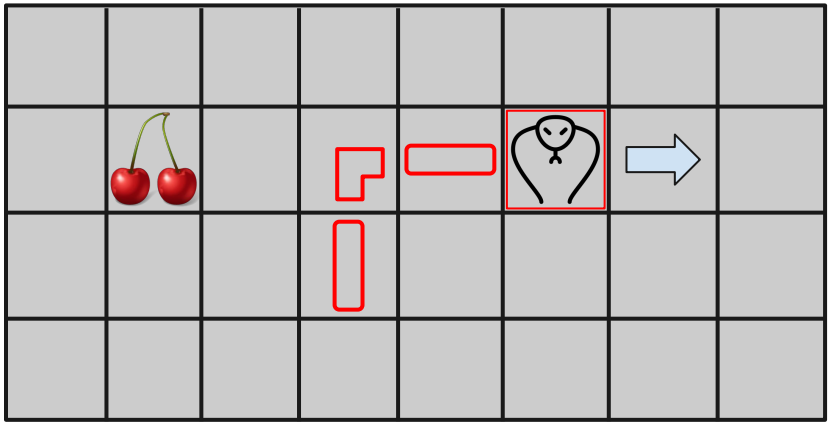
- Khởi tạo: sân chơi, con rắn, vị trí quả
- Game loop, tại mỗi bước:
  - Xử lý sự kiện bàn phím để đổi hướng đi bước tiếp theo
  - Xử lý game logic: di chuyển rắn theo hướng đi hiện tại, va chạm tường, va chạm thân rắn, ăn quả dài thân và tăng điểm số
  - Hiển thị màn hình trò chơi

Nội dung

- Trò chơi: Snake
- **Sân chơi**
  - **Mảng 2 chiều**
- Con rắn
  - Danh sách liên kết có đuôi
- Bắt phím di chuyển rắn
  - SDL_PollEvent()
- Xử lý va chạm

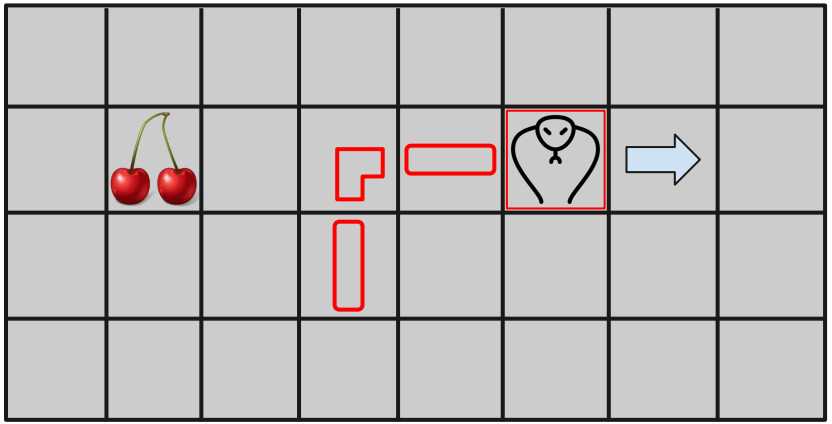
Phân tích trạng thái trò chơi: Sân chơi

- Sân chơi là bảng hình chữ nhật, gồm các ô
  - Ô rỗng
  - Ô có rắn
  - Ô có quả
- Sân chơi còn có
  - Con rắn
    - và hướng đi
  - Quả cherry
    - vị trí cherry



Phân tích trạng thái trò chơi: Sân chơi

- Sân chơi là bảng hình chữ nhật, gồm các ô
  - Ô rỗng
  - Ô có rắn
  - Ô có quả



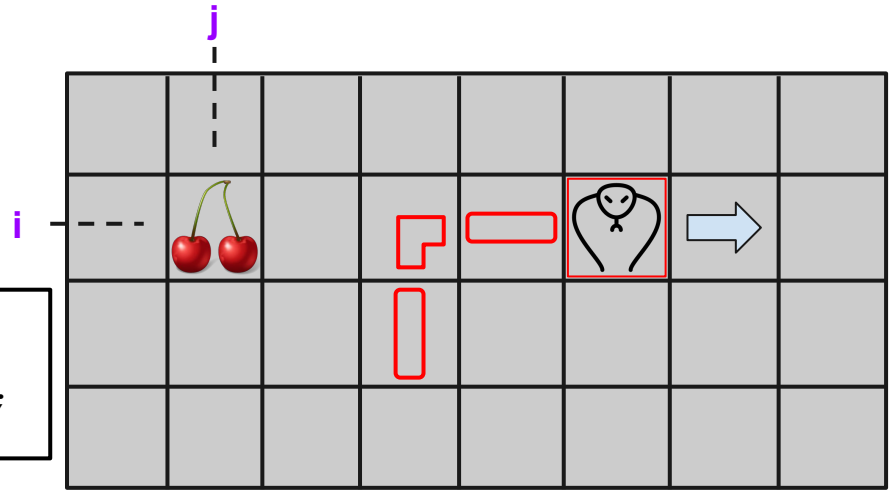
Mô tả các loại ô bằng enum

```
enum CellType {
    CELL_EMPTY = 0,
    CELL_SNAKE,
    CELL_CHERRY
};
```

các loại ô

Phân tích trạng thái trò chơi: Sân chơi

- Sân chơi là bảng hình chữ nhật, gồm các ô
  - Ô rỗng
  - Ô có rắn
  - Ô có quả



Một cách biểu diễn sân chơi

```
std::vector<
 std::vector<CellType> > squares;
```

mỗi dòng là một `vector<CellType>`

một bảng gồm nhiều dòng (vector các vector)

`squares[i][j]`: trạng thái dòng i cột j

lấy phần tử thứ j của vector thứ i của bảng

Phân tích trạng thái trò chơi: Sân chơi

```
std::vector<
 std::vector<CellType> > squares;
```

đủ thông tin để vẽ sân chơi một cách đơn giản bằng cách đánh dấu ô chứa quả và các ô chứa thân rắn

Câu hỏi: để vẽ đầu rắn cần làm gì ?

Đáp: Một phương án là thêm một loại ô, ví dụ CELL_SNAKE_HEAD vào enum CellType, hoặc,

Hỏi sân chơi xem đầu rắn (hoặc toàn bộ thân rắn) ở đâu ?

```
int width;
int height;

// tạo bảng có height dòng, width cột
squares = vector< vector<CellType> > (
    height,
    vector<CellType>(width, CELL_EMPTY)
);

// quét bảng từ trên xuống, từ trái qua
for (int i = 0; i < height; i++) {
    for (int j = 0; j < width; j++) {
        // làm gì đó với squares[i][j]
    }
}
```

Bài tập: Khởi tạo sân chơi

- Bắt đầu tạo lớp sân chơi Game
- Làm hàm khởi tạo 2 tham số: chiều rộng, chiều cao

```
class Game
{
public:
    const int width;
    const int height;

private:
    std::vector< std::vector<CellType> > squares;

public:
    Game(int _width, int _height);
};
```

Bài tập: Thay đổi trạng thái ô

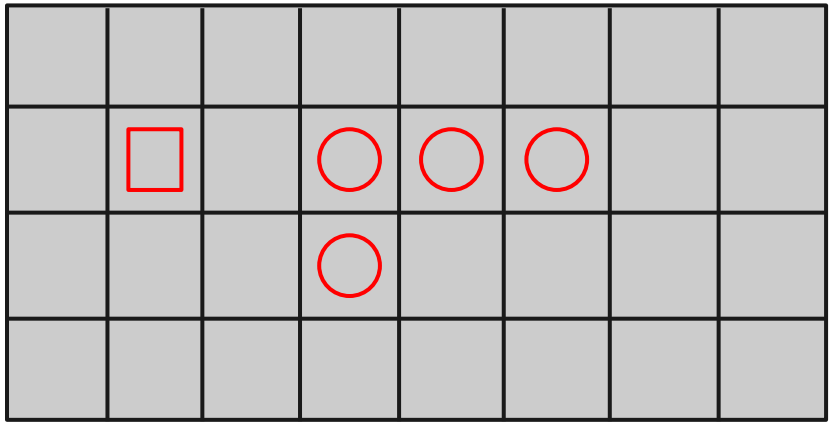
- Viết hàm setCellType(int x, int y, CellType type) thay đổi trạng thái ô tại dòng y, cột x
- Viết hàm addCherry(int x, int y) đặt quả cherry ở dòng y, cột x
- Viết hàm thành viên addRandomCherry() đặt quả cherry ở một vị trí ngẫu nhiên có trạng thái CELL_EMPTY

Bài tập: Vẽ sân chơi đơn giản

- Viết hàm thành viên getSquares() lấy bảng
  - Trả về tham chiếu hằng đến bảng squares
  - Hàm không thay đổi sân chơi (hàm hằng)
- Viết hàm **vẽ sân chơi** *bên ngoài lớp Game*
  - Có tham số là tham chiếu hằng đến Game
  - Vẽ các đường kẻ ngang cách đều nhau
  - Vẽ các đường kẻ dọc
  - Duyệt bảng,
    - nếu ô chứa quả, vẽ hình vuông;
    - nếu ô chứa rắn, vẽ hình tròn.

Bài tập: Vẽ sân chơi đơn giản

Kết quả cần đạt được ở bài tập này

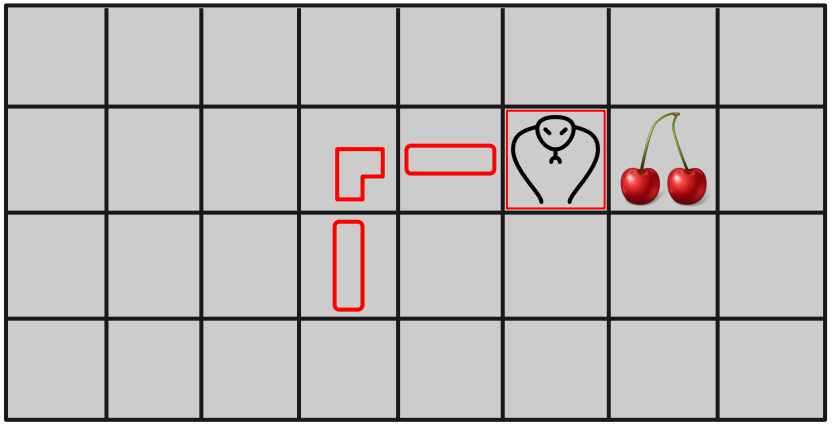


Nội dung

- Trò chơi: Snake
- Sân chơi
  - Mảng 2 chiều
- **Con rắn**
  - **Danh sách liên kết có đuôi**
- Bắt phím di chuyển rắn
  - SDL_PollEvent()
- Xử lý va chạm

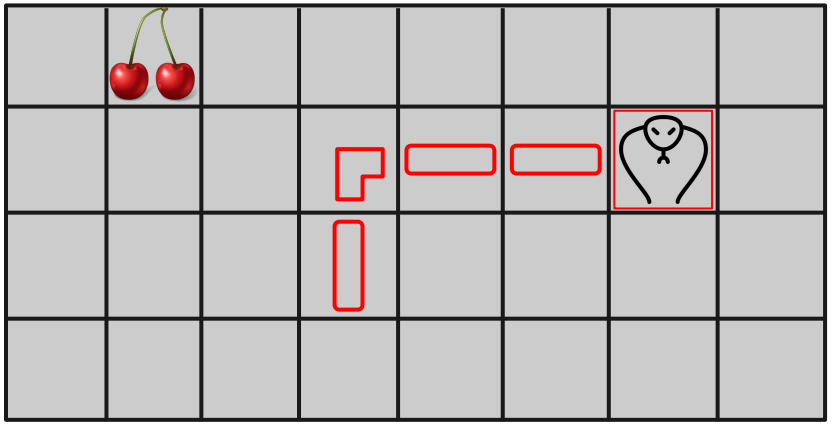
Phân tích trạng thái trò chơi: Con rắn

- Con rắn là một chuỗi vị trí các ô trong bảng
- Di chuyển theo 1 hướng
  - **Ăn quả**
    - **Dài ra**
  - Không ăn quả
    - Vị trí các đốt tịnh tiến



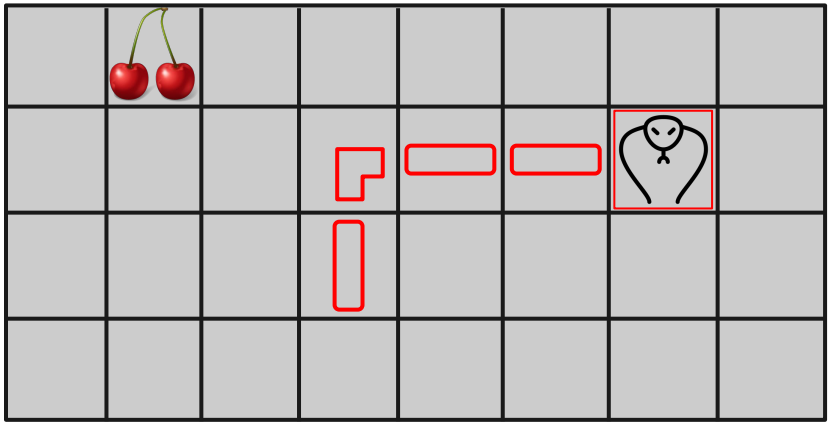
Phân tích trạng thái trò chơi: Con rắn

- Con rắn là một chuỗi vị trí các ô trong bảng
- Di chuyển theo 1 hướng
  - **Ăn quả**
    - **Dài ra**
  - Không ăn quả
    - Vị trí các đốt tịnh tiến



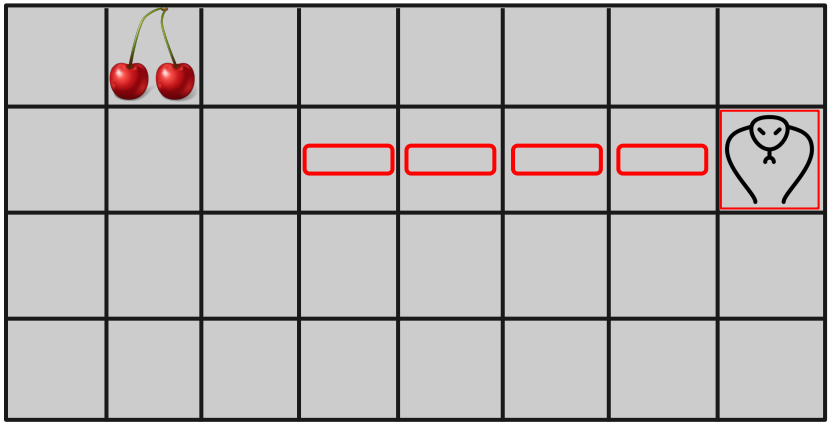
Phân tích trạng thái trò chơi: Con rắn

- Con rắn là một chuỗi vị trí các ô trong bảng
- Di chuyển theo 1 hướng
  - Ăn quả
    - Dài ra
  - **Không ăn quả**
    - **Vị trí các đốt tịnh tiến**



Phân tích trạng thái trò chơi: Con rắn

- Con rắn là một chuỗi vị trí các ô trong bảng
- Di chuyển theo 1 hướng
  - Ăn quả
    - Dài ra
  - **Không ăn quả**
    - **Vị trí các đốt tịnh tiến**

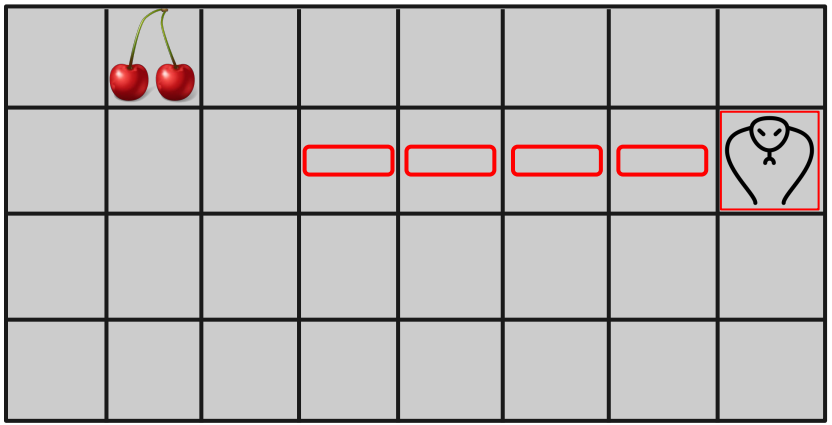


Biểu diễn con rắn

- Con rắn là một chuỗi vị trí các ô trong bảng
- Di chuyển theo 1 hướng nào đó

```
enum Direction {
    UP = 0, DOWN, LEFT, RIGHT
};
```

Dùng enum để mô tả các hướng đi

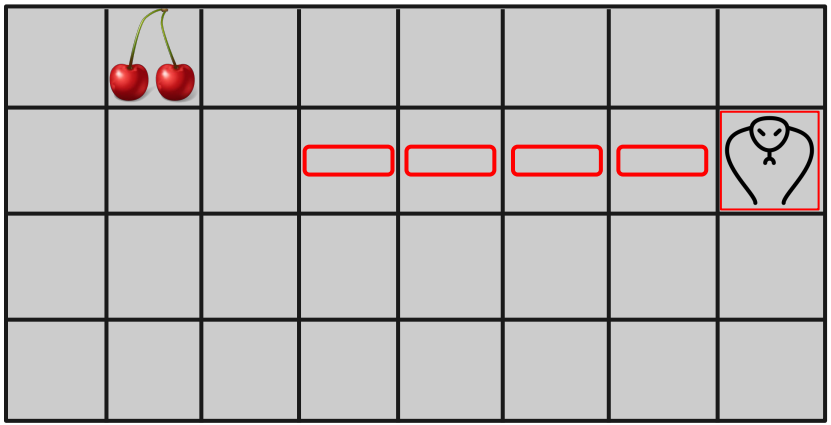


Biểu diễn con rắn

- Con rắn là một chuỗi vị trí các ô trong bảng
- Vị trí gồm tọa độ x, y

```
struct Position
{
    int x;
    int y;

    Position(int x_ = 0, int y_ = 0);
};
```

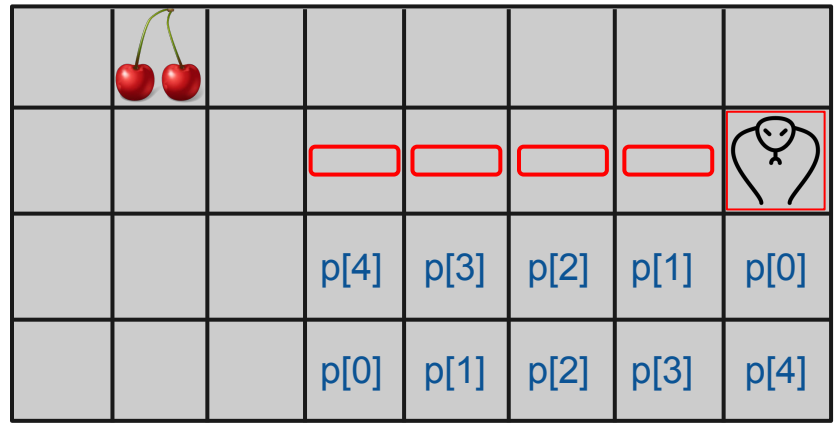


Bài tập: viết hàm khởi tạo một vị trí

Biểu diễn con rắn

- Con rắn là một chuỗi vị trí các ô trong bảng
- Cách 1: sử dụng vector

```
class Snake {
    std::vector<Position> positions;
};
```



Suy nghĩ:

Các chức năng của rắn cần cài đặt thế nào

- Nếu positions[0] là đầu rắn, cần chèn vào đầu vector khi ăn quả (dịch cả vector về sau)
- Nếu positions[0] là đuôi rắn (positions[4] là đầu rắn), ăn quả = push_back
  - Nhưng khi không ăn quả vẫn phải duyệt từ đầu đến cuối con rắn để thay đổi vị trí

Có cách nào hay hơn ?

Tại sao cần cách hiệu quả hơn ?

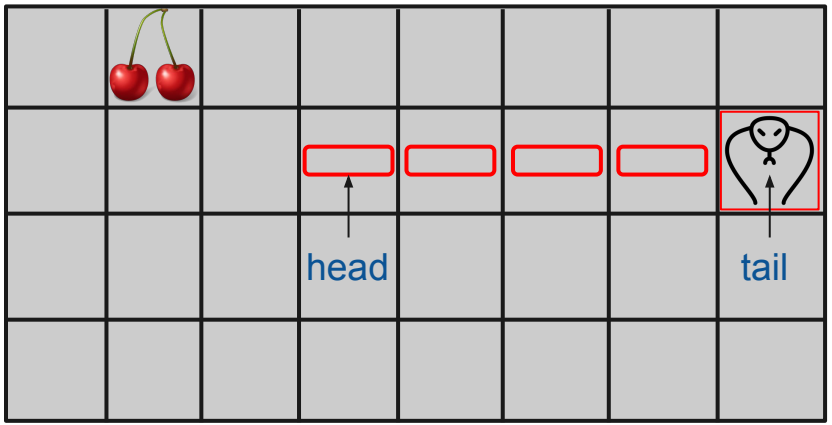
- Khi rắn chỉ có ít đốt
  - Cách cài đặt nào cũng chạy nhanh
- *Khi rắn nhiều đốt* (gần kín màn hình)
  - Nếu cài đặt không đủ nhanh
    - Hình vẽ giật
    - Người chơi có thể lỡ nhịp, thua cuộc
- Các trò chơi càng phức tạp
  - Xử lý logic của game càng phải hiệu quả

Biểu diễn con rắn: Cách hay hơn

- Con rắn là một chuỗi vị trí các ô trong bảng
- Cách 2: sử dụng *danh sách liên kết có đuôi*

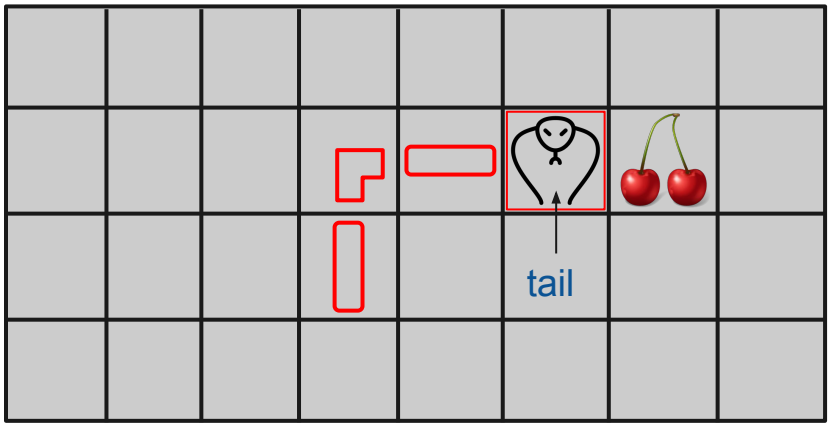
```
struct SnakeNode
{
    Position position;
    SnakeNode  *next;
    SnakeNode(Position p,
              SnakeNode * _next = nullptr)
        : position(p), next(_next) {}
};

class Snake
{
    SnakeNode  *head, *tail;
};
```



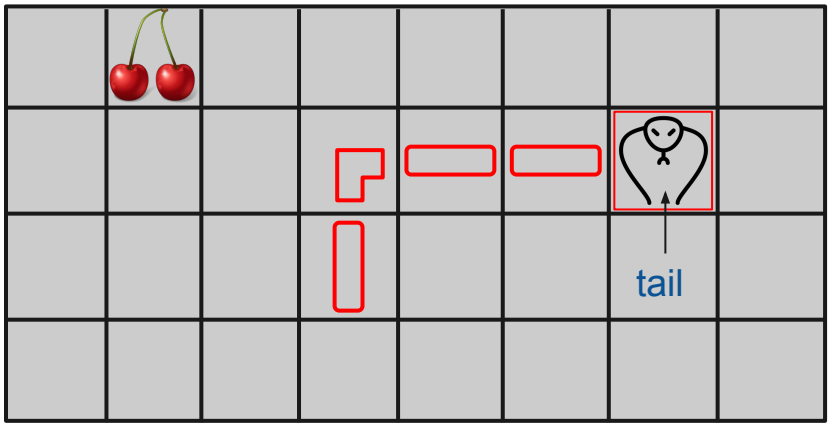
Biểu diễn con rắn: Cách hay hơn

- Con rắn là một chuỗi vị trí các ô trong bảng
- Cách 2: sử dụng danh sách liên kết có đuôi
- **Ăn quả**
  - **Dài ra**



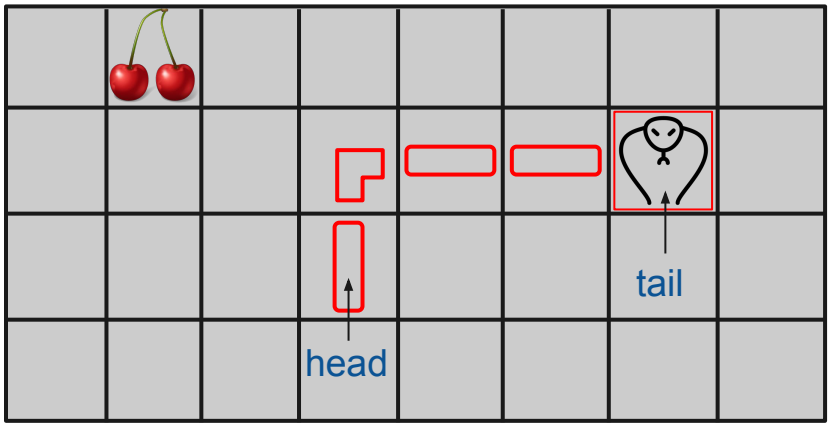
Biểu diễn con rắn: Cách hay hơn

- Con rắn là một chuỗi vị trí các ô trong bảng
- Cách 2: sử dụng danh sách liên kết có đuôi
- **Ăn quả**
  - **Dài ra**
  - **= addLast(newPos)**



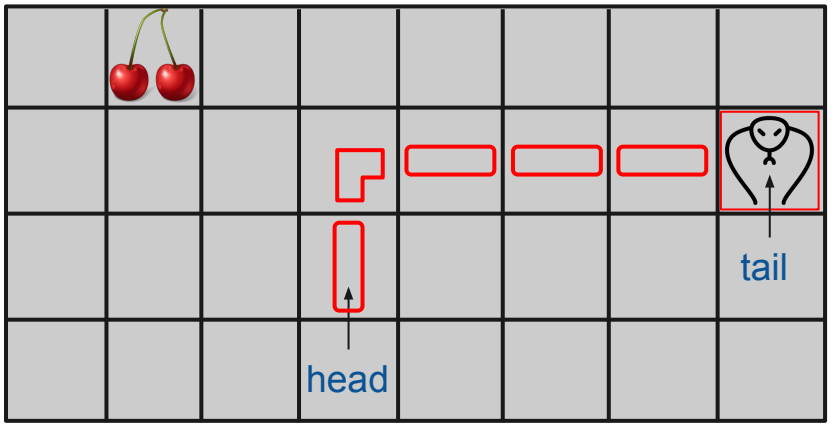
Biểu diễn con rắn: Cách hay hơn

- Con rắn là một chuỗi vị trí các ô trong bảng
- Cách 2: sử dụng danh sách liên kết có đuôi
- **Không ăn quả**
  - **Tịnh tiến các vị trí**



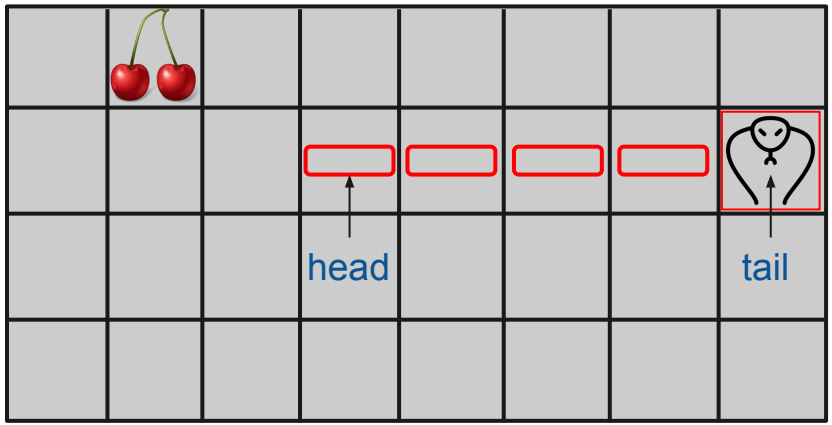
Biểu diễn con rắn: Cách hay hơn

- Con rắn là một chuỗi vị trí các ô trong bảng
- Cách 2: sử dụng danh sách liên kết có đuôi
- **Không ăn quả**
  - **Tịnh tiến các vị trí**
  - **= addLast(newPos)**



Biểu diễn con rắn: Cách hay hơn

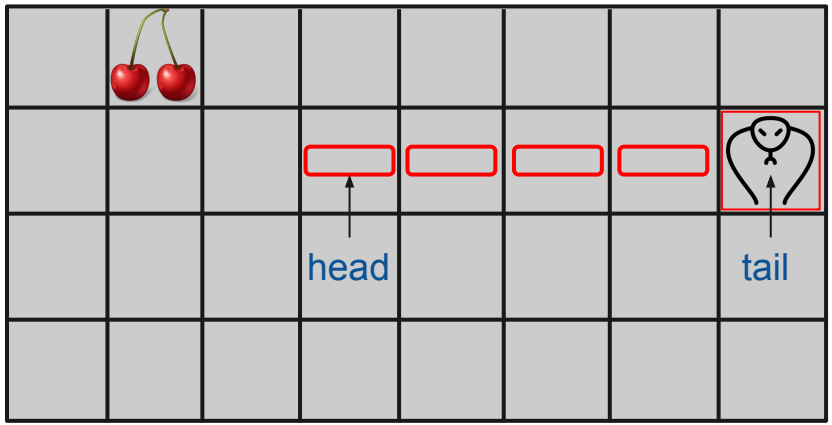
- Con rắn là một chuỗi vị trí các ô trong bảng
- Cách 2: sử dụng danh sách liên kết có đuôi
- **Không ăn quả**
  - **Tịnh tiến các vị trí**
  - **= addLast(newPos)**
    + removeFirst()



- *Cả hai chức năng đều có cài đặt nhanh, hiệu quả*
  - Không cần duyệt và thay đổi vị trí của từng đốt

Biểu diễn con rắn: Cách hay hơn

- Con rắn là một chuỗi vị trí các ô trong bảng
- Cách 2: sử dụng danh sách liên kết có đuôi
- Lưu ý:
  - head là *đầu danh sách liên kết* nhưng trỏ đến *đuôi rắn*
  - tail là *đuôi danh sách liên kết* nhưng trỏ đến *đầu rắn*
  - Câu hỏi: có thể đảo vai trò của head và tail không ?



Bài tập: Lớp Position

- Viết hàm **move(direction)**
  - Trả về vị trí tương ứng khi di chuyển từ một vị trí theo các hướng UP, DOWN, LEFT, RIGHT
- Viết toán tử **==** so sánh 2 vị trí có bằng nhau
- Viết hàm **isInsideBox(left, top, width, height)**
  - Kiểm tra vị trí có nằm trong hình chữ nhật có chiều dài width, chiều cao height, có góc trái trên ở tọa độ (left, top)

Bài tập: Sửa lớp Game

- Sửa các hàm setCellType, addCherry, addRandomCherry sử dụng Position thay cho các tọa độ x, y ở tham số
  - Kiểm tra vị trí có nằm trong hình chữ nhật (0,0,width, height) trước khi thay đổi trạng thái ô
  - Sử dụng hàm kiểm tra isInsideBox(...)

Bài tập: Lớp Snake

- Viết hàm khởi tạo Snake(startPos) có tham số là 1 vị trí (đốt đầu tiên của rắn)
- Viết hàm hủy ~Snake() giải phóng bộ nhớ danh sách liên kết các đốt rắn
- Viết hàm growAtFront(newPos) làm dài rắn ở đầu (tương đương addLast)
- Viết hàm slideTo(newPos) tịnh tiến các vị trí của rắn (tương đương addLast+removeFirst)

Bài tập: Lớp Snake (tiếp)

- Thêm 1 trường int cherry; vào lớp Snake
  - Khởi tạo cherry = 0 trong hàm khởi tạo
- Viết hàm eatCherry(), tăng cherry lên 1
  - Nếu cherry > 0, nghĩa là rắn vừa ăn quả cherry
- Viết hàm move(direction) di chuyển rắn theo hướng direction
  - Tìm vị trí mới qua hàm move(direction) của vị trí đầu rắn
  - Nếu cherry > 0, gọi growAtFront(newPos) rồi giảm cherry
  - Nếu cherry == 0, gọi slideTo(newPos)

Bài tập: Kết nối Game và Snake

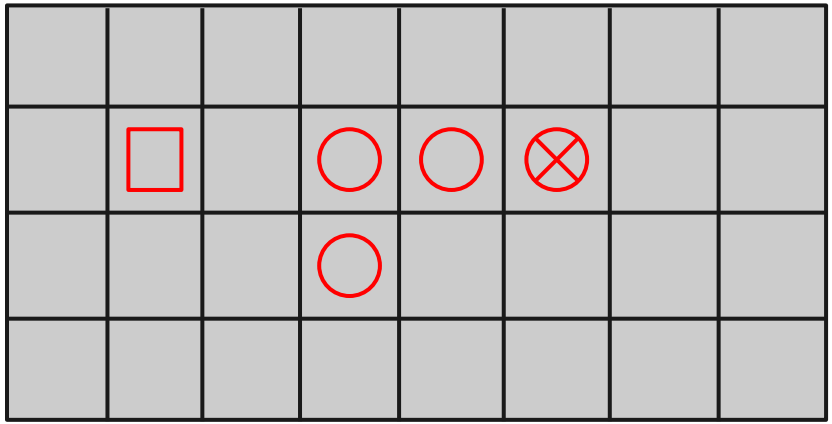
- Game cần chứa thông tin về con rắn
  - Thêm 1 trường Snake snake; vào lớp Game
  - Thêm 1 trường tham chiếu Game& game; vào lớp Snake
  - Sửa hàm khởi tạo lớp Snake thành Snake(Game& game_, Position startPos)
    - Khởi tạo trường tham chiếu game
  - Sửa hàm khởi tạo lớp Game
    - Khởi tạo trường snake với tham số *this và vị trí ở giữa màn hình Position(width/2,height/2)

Bài tập: Sửa lớp Game

- Viết hàm getSnakePositions() trả về vector các vị trí của rắn
  - Viết và gọi hàm getPositions() trong lớp Snake
- Viết hàm getCherryPosition() trả về vị trí cherry
  - Thêm trường cherryPosition
  - Sửa hàm addCherry() để cập nhật trường này
- Sửa hàm vẽ sân chơi để vẽ đầu rắn
  - *Lấy vị trí rắn và vị trí quả cherry để vẽ*

Bài tập: Vẽ đầu rắn

Kết quả cần đạt được ở bài tập này



Bài tập: Kết nối Game và Snake

- Viết hàm snakeMoveTo(pos) thông báo rắn di chuyển đến ô mới
  - Kiểm tra pos nếu là CELL_CHERRY, gọi snake.eatCherry() và addRandomCherry()
  - Trạng thái mới CELL_SNAKE
- Viết hàm snakeLeave(pos) thông báo rắn rời khỏi ô
  - Trạng thái mới CELL_EMPTY

Bài tập: Kết nối Game và Snake

- Thêm trường Direction currentDirection;
- Sửa hàm khởi tạo Game()
  - Gọi addRandomCherry() để khởi tạo quả cherry đầu tiên
  - Ban đầu currentDirection hướng sang phải (RIGHT)
- Sửa hàm khởi tạo Snake()
  - Gọi game.snakeMoveTo(startPos) để khởi tạo trạng thái ô đầu tiên có rắn

Bài tập: Kết nối Game và Snake

- Sửa hàm move(direction) của Snake
  - Trường hợp cherry > 0, chỉ gọi game.snakeMoveTo(newPos)
  - Trường hợp cherry == 0, gọi game.snakeLeave(tailPos) trước khi gọi game.snakeMoveTo(newPos) (tại sao ?)
- *Gợi ý:* rắn có thể di chuyển vào ô có đuôi của mình ở bước trước

Nội dung

- Trò chơi: Snake
- Sân chơi
  - Mảng 2 chiều
- Con rắn
  - Danh sách liên kết có đuôi
- **Bắt phím di chuyển rắn**
  - **SDL_PollEvent()**
- Xử lý va chạm

Game loop

```
auto start = CLOCK_NOW();
renderGamePlay(renderer, game);
while (game.isGameRunning()) {
    while (SDL_PollEvent(&e)) {
        interpretEvent(e, game);
    }

    auto end = CLOCK_NOW();
    ElapsedTime elapsed = end-start;
    if (elapsed.count() > STEP_DELAY) {
        game.nextStep();
        renderGamePlay(renderer, game);
        start = end;
    }
    SDL_Delay(1);
}
```

kiểm tra trò chơi còn tiếp tục ?

thông báo sự kiện cho trò chơi

kiểm tra xem đã đủ thời gian để di chuyển rắn

Đợi 1 milli giây trước khi lặp tiếp, tránh CPU hoạt động quá nóng

Trạng thái trò chơi

```
enum GameStatus {
    GAME_RUNNING = 1,
    GAME_STOP = 2,
    GAME_WON = 4 | GAME_STOP, // GAME_WON tức là GAME_STOP
    GAME_OVER = 8 | GAME_STOP, // tương tự cho GAME_OVER
};
```

Bài tập:

- Thêm trường status vào lớp Game
- Khởi tạo status là GAME_RUNNING
- Viết các hàm isGameRunning, isGameOver

Thông báo sự kiện phím

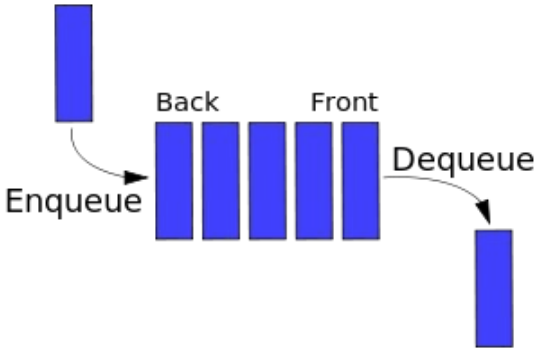
```
void interpretEvent(SDL_Event e, Game & game)
{
    if (e.type == SDL_KEYUP) {
        switch (e.key.keysym.sym) {
            case SDLK_UP: game.processUserInput(UP);  break;
            case SDLK_DOWN: game.processUserInput(DOWN);  break;
            case SDLK_LEFT: game.processUserInput(LEFT);  break;
            case SDLK_RIGHT: game.processUserInput(RIGHT);  break;
        }
    }
}
```

Truyền hướng đi mới vào trong game, thông qua hàm processUserInput()

Thông báo sự kiện phím

- Hàm processUserInput(direction)
  - Chỉ làm nhiệm vụ lưu trữ các yêu cầu di chuyển của người chơi
  - Người chơi có thể nhấn nhiều phím liên tục
    - Lưu trữ các hướng đi trong trường hàng đợi std::queue<Direction> inputQueue;

```
void Game::processUserInput(
    Direction direction )
{
    inputQueue.push(direction);
}
```



Hàng đợi là cấu trúc giúp dữ liệu được lấy lần lượt theo thứ tự xuất hiện (vào trước ra trước - FIFO)

Di chuyển rắn

- Hàm nextStep()
  - Lần lượt lấy các hướng trong inputQueue đến khi chọn được hướng phù hợp hoặc hết hàng đợi
  - Kiểm tra xem có hợp lệ
    - Ví dụ: đang sang phải thì chỉ rẽ lên hoặc xuống
  - Nếu hợp lệ thì thay đổi currentDirection
  - Di chuyển rắn, gọi snake.move(currentDirection);

Di chuyển rắn

- Hàm nextStep()

```
void Game::nextStep()
{
    while (!inputQueue.empty()) {
        Direction next = inputQueue.front();
        inputQueue.pop();
        if (canChange(currentDirection, next))
{
            currentDirection = next;
            break;
        }
    }
    snake.move(currentDirection);
}
```

```
bool Game::canChange(
        Direction current,
        Direction next) const
{
    if (current == UP ||
        current == DOWN)
        return next == LEFT ||
               next == RIGHT;
    else
        return next == UP ||
               next == DOWN;
}
```

Nội dung

- Trò chơi: Snake
- Sân chơi
  - Mảng 2 chiều
- Con rắn
  - Danh sách liên kết có đuôi
- Bắt phím di chuyển rắn
  - SDL_PollEvent()
- **Xử lý va chạm**

Xử lý va chạm

- Có 2 kiểu va chạm:
  - Chạm tường
    - Vị trí mới ngoài hình chữ nhật (0,0,width,height)
  - Chạm thân rắn
    - Vị trí mới có trạng thái CELL_SNAKE
- Khi rắn di chuyển, nó thông báo với Game thông qua hàm snakeMoveTo(newPos)
  - Có thể kiểm tra, xử lý va chạm ở hàm này

Xử lý va chạm

```
void Game::snakeMoveTo(Position pos) {
    if (squares[pos.y][pos.x] == CELL_CHERRY) {
        snake.eatCherry();
        addRandomCherry();
    }
    setCellType(pos, CELL_SNAKE);
}
```

Xử lý va chạm

```
void Game::snakeMoveTo(Position pos) {
    if (squares[pos.y][pos.x] == CELL_CHERRY) {
        snake.eatCherry();
        addCherry();
    }
    setCell
}
```

```
void Game::snakeMoveTo(Position pos) {
    if (!pos.isInsideBox(0,0,width,height)) {
        status = GAME_OVER;
    } else if (squares[pos.y][pos.x] == CELL_SNAKE) {
        status = GAME_OVER;
    } else if (squares[pos.y][pos.x] == CELL_CHERRY) {
        snake.eatCherry();
        addRandomCherry();
        setCellType(pos, CELL_SNAKE);
    } else setCellType(pos, CELL_SNAKE);
}
```

Xử lý va chạm: cách cài đặt đẹp hơn

```
void Game::snakeMoveTo(Position pos) {
    if (!pos.isInsideBox(0,0,width,height)) {
        status = GAME_OVER;
    } else if (squares[pos.y][pos.x] == CELL_SNAKE) {
        status = GAME_OVER;
    } else if (squares[pos.y][pos.x] == CELL_CHERRY) {
        snake.eatCherry();
        addRandomCherry();
        setCellType(pos, CELL_SNAKE);
    } else setCellType(pos, CELL_SNAKE);
}
```

Xử lý va chạm: cách cài đặt đẹp hơn

```
void Game::snakeMoveTo(Position pos) {
    if (!pos.isInsideBox(0,0,width,height)) {
        status = GAME_OVER;
    } else if (squares[pos.y]
        status = GAME_OVER;
    } else if (squares[pos.y]
        snake.eatCherry();
        addRandomCherry();
        setCellType(pos, CELL_
    } else setCellType(pos, C
}
```

```
void Game::snakeMoveTo(Position pos) {
    switch(getCellType(pos)) {
        case CELL_OFF_BOARD:
        case CELL_SNAKE:
            status = GAME_OVER;
            break;
        case CELL_CHERRY:
            snake.eatCherry();
            addRandomCherry();
        default:
            setCellType(pos, CELL_SNAKE);
    }
}
```

Xử lý va chạm: cách cài đặt đẹp hơn

```
CellType Game::getCellType(Position pos) const
{
    return pos.isInsideBox(0, 0, width, height) ?
        squares[pos.y][pos.x] : CELL_OFF_BOARD;
}
```

- Thêm một loại ô CELL_OFF_BOARD vào enum CellType để thể hiện một vị trí nằm ngoài sân chơi
- Kiểm tra game.isGameOver() trong Snake::move() khi gọi game.snakeMoveTo()

Tổng kết

- *Mảng hai chiều*: dữ liệu dạng bảng
- *Danh sách liên kết*: dữ liệu dạng chuỗi cần chèn, xóa nhanh
- *Hàng đợi*: lưu trữ yêu cầu của người chơi theo thứ tự xuất hiện (vào trước ra trước - FIFO)
- *Định nghĩa enum*: xử lý nhiều trường hợp một cách thống nhất (và đặt tên cho chúng)

Bài tập

- Làm đẹp cách thể hiện bằng cách nạp ảnh vẽ đầu rắn, thân rắn, các đốt rắn chuyển hướng
- Gợi ý:
  - Cài đặt lớp Gallery chuyên quản lý các hình vẽ
  - Truy xuất các hình vẽ bằng enum
    - Đặt tên cho hình vẽ
  - Xét các trường hợp để vẽ thân rắn
    - Cần xét vị trí tương quan của 3 đốt liên tiếp

Bài tập

- Lưu điểm số (số quả cherry ăn được)
- Lưu bảng xếp hạng điểm số
- Vẽ màn hình khởi động
  - Tên người chơi nhập từ tham số dòng lệnh
  - Nhấn Enter để chơi
  - Nhấn R để xem bảng xếp hạng
- Vẽ màn hình kết thúc
  - Bảng xếp hạng, làm nổi bật điểm số của lần chơi vừa kết thúc