

# Snake Game

13&14 - Danh sách liên kết

# Nội dung

- Trò chơi: Snake
- Kỹ thuật
  - Mảng 2 chiều
  - Bắt phím với `SDL_PollEvent()`
  - Hàng đợi
    - xử lý hiện tượng rớt phím
  - Danh sách liên kết
    - thêm, chèn, xóa trên danh sách hiệu quả

# Trò chơi Snake

- Sân chơi hình chữ nhật
  - Trên sân chơi xuất hiện các quả cherry ngẫu nhiên
- Rắn lúc đầu
  - dài 4 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=kTIPpblbkos>

# Demo - Start Screen

# Demo - Midgame screen

# Các tác vụ của trò chơi

- **Hiển thị hình vẽ giới thiệu**
  - Có nút hiển thị bảng xếp hạng các lần 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

# Lộ trình xây dựng trò chơi

Các phiên bản

0.1: vẽ sân chơi và rắn đơn giản (dùng ô vuông hoặc hình tròn), điều khiển được rắn di chuyển

0.2: thêm quả vào sân chơi, rắn ăn quả dài ra

0.3: xử lý va chạm với cạnh sân và thân rắn

0.4: Vẽ các đốt rắn đẹp bằng ảnh JPG

1.0: Thêm màn hình khởi động, điểm số, bảng xếp hạng

# Chuẩn bị

- Tạo project Snake
- Cài đặt thư viện **SDL2, SDL2\_image**
- Đưa **main.cpp, painter.h, painter.cpp** từ bài giảng về SDL vào project
- Sửa **main.cpp**
  - Xoá các hàm vẽ
  - Sửa tiêu đề cửa sổ
  - Chỉ để lại mã khởi tạo và giải phóng SDL
    - cửa sổ và bút vẽ



# Chuẩn bị

## Hàm `main()`

```
int main(int argc, char* argv[])
{
    srand(time(0));
    SDL_Window* window;
    SDL_Renderer* renderer;
    initSDL(window, renderer);
    Painter painter(window, renderer);

    // TODO: game code here

    quitSDL(window, renderer);
    return 0;
}
```

# Mã giả

```
render splash screen;
initialize play-ground size = (width, height)

render play-ground (save timestamp)
while (game is running) {

    get user input
    update snake direction using user input (turn up, down, left, right)

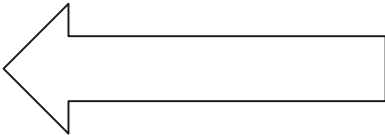
    if elapsed time > required delay between steps
        move the game (snake crawl, generate cherry) to the next step
        render play-ground
        save new timestamp
}
render game-over screen
update score and ranking table to file
```

# Code C++

```
renderSplashScreen();
PlayGround playGround(GROUND_WIDTH, GROUND_HEIGHT);
SDL_Event e;
renderGamePlay(painter, playGround);
auto start = CLOCK_NOW();
while (playGround.isGameRunning()) {
    while (SDL_PollEvent(&e) != 0) {
        UserInput input = interpretEvent(e);
        playGround.processUserInput(input);
    } // non-blocking event detection

    // game logic here

    SDL_Delay(1); // to prevent high CPU usage because of SDL_PollEvent()
}
renderGameOver(painter, playGround);
updateRankingTable(playGround);
```



```
auto end = CLOCK_NOW();
ElapsedTime elapsed = end-start;
if (elapsed.count() > STEP_DELAY)
{
    playGround.nextStep();
    renderGamePlay(painter,
                    playGround);

    start = end;
}
```

# Một số tiện ích

```
// số giây giữa hai lần vẽ
```

```
const double STEP_DELAY = 0.5;
```

```
// tên ngắn của hàm lấy thời gian
```

```
#define CLOCK_NOW chrono::system_clock::now
```

```
// Kiểu đại diện cho khoảng thời gian (tính theo giây)
```

```
typedef chrono::duration<double> ElapsedTime;
```

# Nhập liệu và hiển thị

```
const int GROUND_WIDTH = 30;  
const int GROUND_HEIGHT = 20;
```

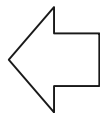
```
void renderSplashScreen();  
void renderGamePlay(Painter&, const PlayGround& playGround);  
void renderGameOver(Painter&, const PlayGround& playGround);  
UserInput interpretEvent(SDL_Event e);  
void updateRankingTable(const PlayGround& playGround);
```

- PlayGround: lớp biểu diễn sân chơi
  - Xử lý logic của game
- UserInput: các hành động của người chơi

```
enum UserInput { NO_INPUT = 0, KEY_UP, KEY_DOWN, KEY_LEFT, KEY_RIGHT };
```

# Tạo các hàm rỗng để lấy chỗ

```
void renderSplashScreen()  
{  
    waitUntilKeyPressed();  
}
```



Đợi 1 phím trước  
khi bắt đầu chơi

```
void renderGamePlay(Painter&, const PlayGround& playGround) { }
```

```
void renderGameOver(Painter&, const PlayGround& playGround) { }
```

```
UserInput interpretEvent(SDL_Event e)  
{  
    return NO_INPUT;  
}
```

```
void updateRankingTable(const PlayGround& playGround) { }
```

# Biểu diễn sân chơi

- Tìm cách biểu diễn mỗi đối tượng trong trò chơi bằng Lớp (dữ liệu + hàm)
- Sân chơi
  - Hình chữ nhật các ô vuông
  - Mỗi ô có thể trống, vị trí của rắn, vị trí của quả
    - Có thể mở rộng sau này để có nhiều loại quả
  - Các chức năng chính (mình có thể nghĩ ra bây giờ)
    - Khởi tạo (và các Getters đọc trạng thái)
    - Thêm quả vào chỗ trống
    - Thay đổi trạng thái các ô

# Biểu diễn sân chơi (PlayGround.\*)

- Enum loại ô trong sân

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

- Dữ liệu của lớp PlayGround

- Hình chữ nhật → mảng 2 chiều trạng thái

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

- Con rắn

```
Snake snake;
```

- Cần tạo lớp Snake

- tạo lớp rỗng trong **Snake.\***

- #include trong **PlayGround.h** để tạm đây

- Điểm số: 

```
int score;
```



# Biểu diễn sân chơi (PlayGround.\*)

- Trạng thái trò chơi: sử dụng các bít 0, 1, 2, 3

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

- Trong lớp PlayGround

```
GameStatus status;  
...  
public:  
...  
    bool isGameRunning() const { return status == GAME_RUNNING; }  
    void processUserInput(UserInput input) { }  
    void nextStep() { }
```

Đến đây chương trình dịch  
được và ta đã lên được khung  
chương trình

# Thay đổi trạng thái ô vuông

- Có thể khai báo

```
void changeCellState(int x, int y, CELL_TYPE type);
```

- Một vị trí luôn có cả 2 biến x và y
- Tạo một **struct Position** để tiện quản lý
  - Sẽ có các hàm thay đổi, so sánh, tính toán vị trí

```
struct Position
```

```
{
```

```
    int x, y;
```

```
};
```

```
class Playground {
```

```
...
```

```
    void changeCellState(Position pos, CellType type);
```

```
};
```

# Khởi tạo sân chơi

- Khởi tạo ô vuông: dựa vào số dòng, số cột
- Khởi tạo rắn
- Thêm 1 quả cherry

```
PlayGround::PlayGround(int width, int height)
: squares(height, vector<CellType>(width, CELL_EMPTY)),
  snake(this), // rắn phụ thuộc vào sân chơi, sửa hàm khởi tạo Snake
  status(GAME_RUNNING),
  score(0)
{
    addCherry();    // thêm 1 hàm đặt squares[0][0] = CELL_CHERRY
                   // để thử nghiệm
}
```

# Sửa hàm khởi tạo Snake

- Cần sửa hàm khởi tạo Snake thành

```
Snake(PlayGround* playGround);
```

- Như vậy,

- trong PlayGround.h có include Snake.h
- trong Snake.h lại include PlayGround.h
- cái nào trước, cái nào sau ? có lỗi ?

- Giải pháp: forward declaration

<http://stackoverflow.com/questions/4757565/what-are-forward-declarations-in-c>

- Khai báo **class PlayGround**; trước khai báo lớp Snake và #include "PlayGround.h" trong Snake.cpp

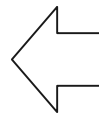
# Phiên bản 0.1

- **Hiển thị đơn giản**
  - Sân chơi: nền tím, ô vuông kẻ màu trắng
  - Rắn: chỉ có 1 đốt hình tròn màu đỏ
  - Quả cherry: hình vuông nhỏ màu cam
- **Điều khiển bằng phím**
  - Lúc đầu rắn ở giữa sân chơi, chạy sang phải
  - Nhận phím mũi tên, chỉnh hướng đi của rắn

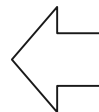
# renderGamePlay(): vẽ sân chơi

```
void renderGamePlay(Painter& painter, const Playground& playGround)
```

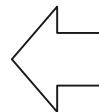
```
{  
    int top = 0, left = 0;  
    int width = playGround.getWidth();  
    int height = playGround.getHeight();  
    painter.clearWithBgColor(PURPLE_COLOR);  
    painter.setColor(WHITE_COLOR);  
    for (int i = 0; i <= width; i++) {  
        painter.setAngle(-90);  
        painter.setPosition(left+i * CELL_SIZE, top+0);  
        painter.moveForward(height * CELL_SIZE);  
    }  
    for (int i = 0; i <= height; i++) {  
        painter.setAngle(0);  
        painter.setPosition(left+0, top+i * CELL_SIZE);  
        painter.moveForward(width * CELL_SIZE);  
    }  
}
```



Vẽ hình tương đối  
với điểm (top, left)



Các đường kẻ dọc



Các đường kẻ ngang

# renderGamePlay(): continue

```
const vector<vector<CellType> >& squares = playGround.getSquares();
for (int i = 0; i < height; i++) {
    for (int j = 0; j < width; j++) {
        if (squares[i][j] == CELL_CHERRY) {
            painter.setColor(ORANGE_COLOR);
            painter.setAngle(-90);
            painter.setPosition(left+j*CELL_SIZE+5, top+i*CELL_SIZE+5);
            painter.createSquare(CELL_SIZE-10);
        } else if (squares[i][j] == CELL_SNAKE) {
            painter.setColor(RED_COLOR);
            painter.setAngle(0);
            painter.setPosition(left+j*CELL_SIZE+5, top+i*CELL_SIZE+CELL_SIZE/2);
            painter.createCircle(CELL_SIZE/2-5);
        }
    }
}
SDL_RenderPresent(painter.getRenderer());
}
```

← Duyệt mảng 2 chiều

← Tìm ô có cherry

← Tìm các đốt rắn

# Biểu diễn con rắn

- Dữ liệu của Snake

- Position position;
- PlayGround\* playGround;

```
struct Position
{
    int x;
    int y;
    Position(int x_, int y_) : x(x_), y(y_) {}
};
```

- Đưa khai báo **struct Position** sang Position.h
  - thêm hàm khởi tạo bằng 2 toạ độ
- #include Position.h trong Snake.h và PlayGround.h



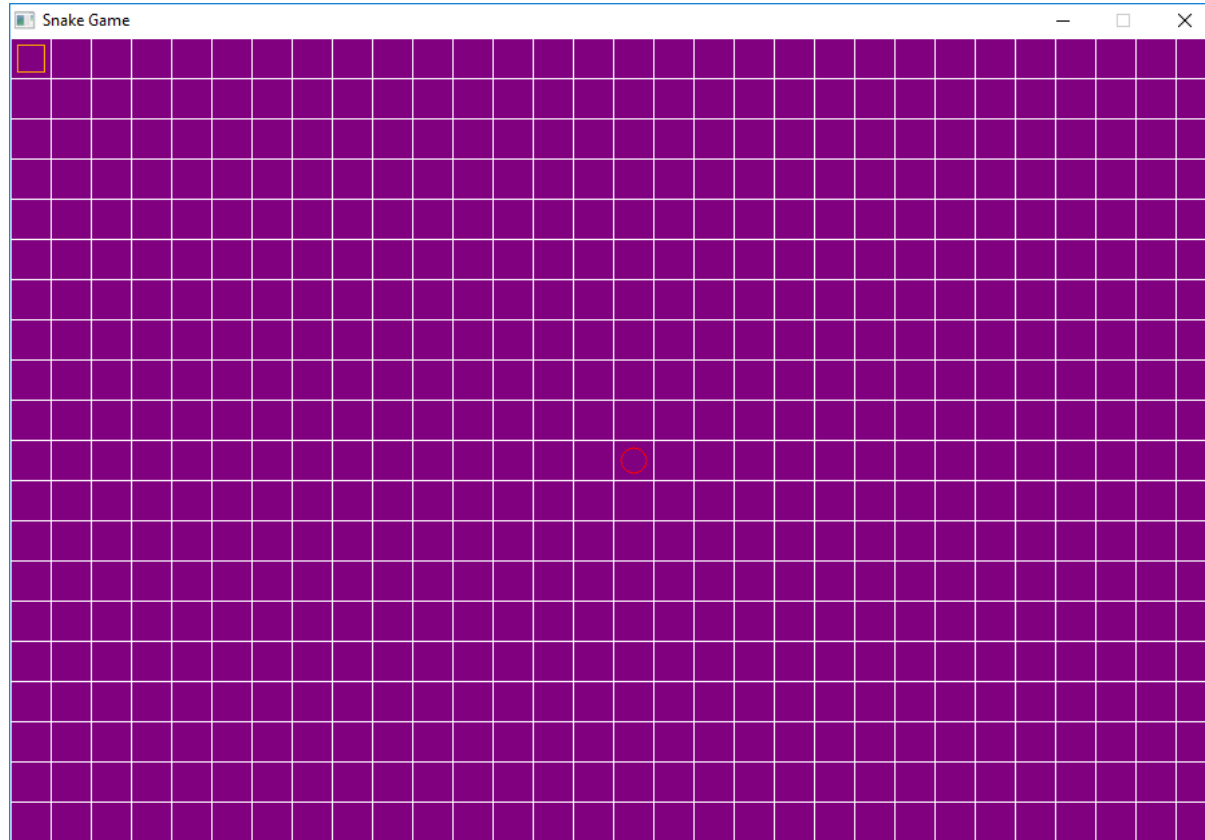
# Khởi tạo rắn

- Khởi tạo đôt ở giữa sân chơi
- Thay đổi trạng thái ở ô này: **CELL\_SNAKE**

```
Snake::Snake(PlayGround* playGround)
    : position(playGround-&gtgetWidth() / 2, playGround-&gtgetHeight() / 2),
      this->playGround(playGround)
{
    playGround->changeCellState(position, CELL_SNAKE);
}
```

```
void PlayGround::changeCellState(Position pos, CellType type)
{
    squares[pos.y][pos.x] = type;
}
```

# Phiên bản 0.1: phần hiển thị



# Phần điều khiển

- Cần chuyển `SDL_Event` thành `UserInput`
  - Hàm `UserInput` `interpretEvent(SDL_Event e);`
- Gọi `Snake.processUserInput()` từ `PlayGround.processUserInput()`
- Thay đổi hướng hiện thời của Snake
  - Thêm dữ liệu vào Snake: `Direction direction;`
- Gọi `Snake.nextStep()` từ `PlayGround.nextStep()`

# Phần điều khiển

- Tạm chuyển khai báo UserInput qua Snake.h
- Khai báo Direction trong Position.h
- Khởi tạo **direction** của Snake là RIGHT
- Tạo các hàm processUserInput, nextStep trong Snake (giống PlayGround)

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

# Tính hướng đi mới của rắn

```
void Snake::processUserInput(UserInput input)
{
    direction = changeDirection(input);
}
```

```
Direction Snake::changeDirection(UserInput input)
```

```
{
    switch (input) {
        case KEY_UP:    return direction != DOWN ? UP : direction;
        case KEY_DOWN:  return direction != UP ? DOWN : direction;
        case KEY_LEFT:  return direction != RIGHT ? LEFT : direction;
        case KEY_RIGHT: return direction != LEFT ? RIGHT : direction;
        default:        return direction;
    }
}
```

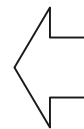


Kiểm tra xem có được  
phép đổi hướng  
(không được đổi  
hướng ngược lại  
hướng đang đi)

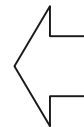
# Di chuyển con rắn

```
void Snake::nextStep()
```

```
{  
    Position newPosition = position.move(direction);  
    playGround->changeCellState(position, CELL_EMPTY);  
    position = newPosition;  
    playGround->changeCellState(position, CELL_SNAKE);  
}
```



Gọi phương  
thức move()  
của Position



Xoá trạng thái ô  
cũ và đặt trạng  
thái ô mới

```
Position Position::move(Direction d)  
{  
    const int dx[] = {0,0,-1,1};  
    const int dy[] = {-1,1,0,0};  
    return Position(x+dx[d],y+dy[d]);  
}
```

# Bắt phím

<https://www.libsdl.org/release/SDL-1.2.15/docs/html/guideinputkeyboard.html>

```
UserInput interpretEvent(SDL_Event e)
{
    if (e.type == SDL_KEYUP) {
        switch (e.key.keysym.sym) {
            case SDLK_UP: return KEY_UP;
            case SDLK_DOWN: return KEY_DOWN;
            case SDLK_LEFT: return KEY_LEFT;
            case SDLK_RIGHT: return KEY_RIGHT;
        }
    }
    return NO_INPUT;
}
```

# Chạy thử

- Đã điều khiển được rắn chạy
- Nhưng
  - Có hiện tượng rớt phím nếu ấn quá nhanh
  - Khi rắn ra ngoài màn hình sẽ bị lỗi Runtime
    - Do ghi trạng thái vào ô nằm ngoài mảng 2 chiều



# Bắt lỗi

```
#include <cassert>
```

- Thêm câu lệnh

```
assert(pos.isInsideBox(0,0,getWidth(),getHeight()));
```

vào hàm `PlayGround::changeCellState()`

- Thêm hàm `Position::isInsideBox(left,top,w,h)` vào lớp `Position`
- Cách này chưa xử lý hết lỗi nhưng cho ta biết lỗi xảy ra là lỗi gì

# Xử lý hiện tượng rớt phím

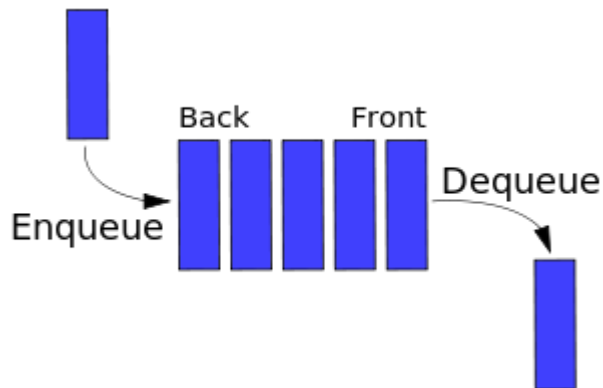
- Nguyên nhân
  - Nếu ấn nhiều phím trong khoảng thời gian giữa 2 lần vẽ, chỉ phím cuối cùng được xử lý
- Cách xử lý:
  - Snake::processUserInput() lưu trữ lại UserInput trong *hàng đợi*
  - Snake::nextStep() lần lượt lấy các UserInput đang chờ ra đến khi
    - Hoạch hết hàng đợi, hoặc
    - Lấy được 1 UserInput có thể thay đổi hướng đi

# Xử lý hiện tượng rớt phím

- Thêm hàng đợi UserInput vào Snake

```
#include <queue>
...
class Snake {
...
    std::queue<UserInput> inputQueue;
...
};
```

```
void Snake::processUserInput(UserInput input)
{
    inputQueue.push(input);
}
```

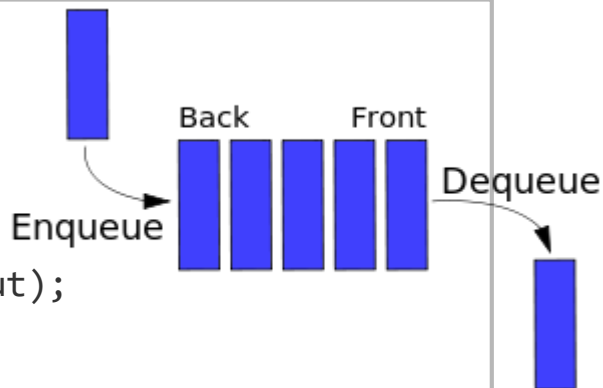


Hàng đợi là cấu trúc giúp dữ liệu được lấy lần lượt theo thứ tự xuất hiện

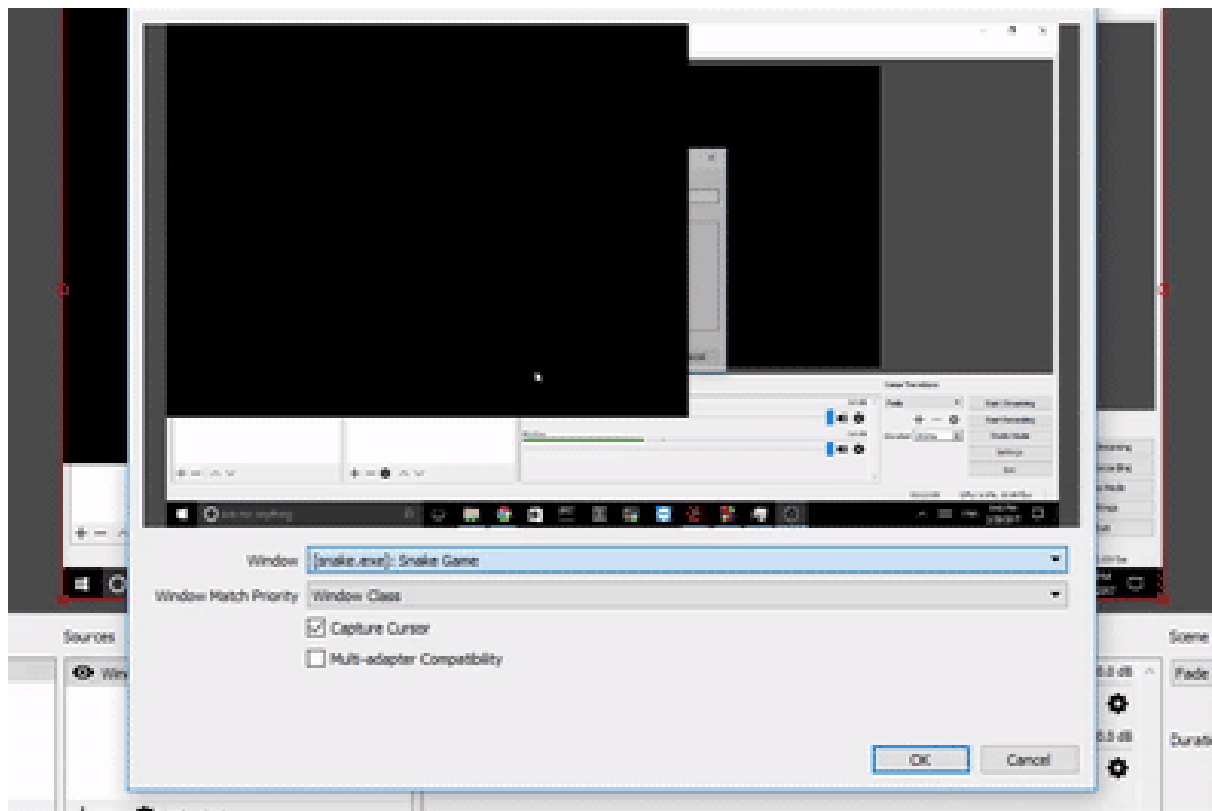
# Xử lý hiện tượng rớt phím

- Thêm hàng đợi UserInput vào Snake

```
void Snake::nextStep()
{
    while (!inputQueue.empty()) {
        UserInput input = inputQueue.front();
        inputQueue.pop();
        Direction newDirection = changeDirection(input);
        if (newDirection != direction) {
            direction = newDirection;
            break;
        }
    }
    Position newPosition = position.move(direction);
    ...
}
```



# Phiên bản 0.1: demo



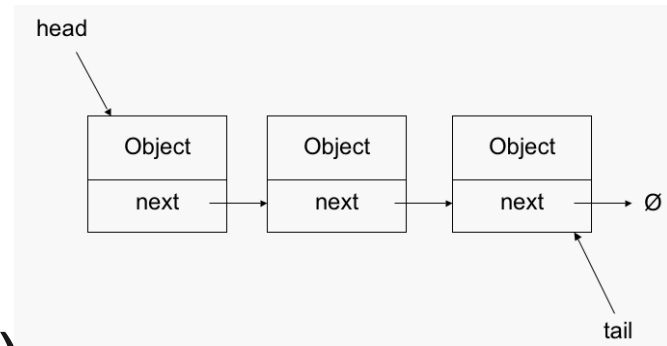
<https://github.com/tqlong/advprogram/archive/2b1981c697c41e5365d5299ab3e966aabe6b6e35.zip>

# Phiên bản 0.2: rắn ăn quả dài ra

- Cần phát hiện ô có quả khi rắn di chuyển (hàm Snake::nextStep())
- Cần lưu trữ nhiều Position cho các đốt rắn
- Khi rắn ăn quả thì bước sau sẽ dài ra:
  - Các đốt cũ giữ nguyên
  - Dài ra bằng cách thêm 1 đốt đầu rắn ở vị trí mới (newPosition)
  - Nếu lưu các đốt ở dạng vector → sẽ phải chèn vào đầu vector → không hiệu quả

# Danh sách liên kết

Là cấu trúc dữ liệu cho phép *chèn, xoá các vị trí trong dãy hiệu quả* (không phải dịch chuyển các phần tử phía sau)



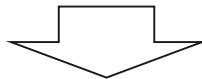
- Mỗi nốt (đốt) có dữ liệu và 1 con trỏ
- Con trỏ sẽ trỏ đến địa chỉ của nốt tiếp theo
- Con trỏ đóng vai trò mối nối giữa các nốt
- Một con trỏ **head** trỏ đến nốt đầu tiên
- Con trỏ của nốt cuối trỏ đến NULL (hết dãy)

# Một đốt của rắn

```
struct SnakeNode
```

```
{  
    Position position;  
    SnakeNode* next;
```

Tạo một đốt mới có dữ liệu p và  
nối tới một đốt cũ



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

## Cách dùng

```
head = nullptr; // danh sách rỗng  
head = new SnakeNode( Position(0,0), head ); // thêm 1 đốt ở (0,0)  
head = new SnakeNode( Position(0,1), head ); // thêm 1 đốt ở (0,1) vào đầu  
head = new SnakeNode( Position(0,2), head ); // thêm 1 đốt ở (0,2) vào đầu
```



# Khởi tạo rắn 1 đốt

- Xoá dữ liệu position trong Snake  
Thay bằng SnakeNode\* head
- Thay lệnh khởi tạo position( ... ) bằng lệnh

```
head( new SnakeNode (  
    Position(playGround-&gtgetWidth() / 2, playGround-&gtgetHeight() / 2)  
) )
```

- Thay các vị trí có position bằng head->position
- Chương trình vẫn chạy như cũ

# Thay đổi trạng thái sân chơi

- Do rắn có thể có nhiều đốt
  - Cần tạo hàm thay đổi trạng thái sân chơi
  - Thay cho câu lệnh `PlayGround::changeCellState()`
- Cần duyệt qua tất cả các đốt rắn
- Thay các lời gọi đến `PlayGround::changeCellState()`
- Chuyển enum `CellType` qua `Snake.h`

```
void Snake::changePlayGroundState(CellType type)
{
    for (SnakeNode* p = head; p != nullptr; p = p->next)
        playGround->changeCellState(p->position, type);
}
```

# Ăn cherry

- Khi ăn cherry, bước sau mới dài thân
  - Cần lưu lại trạng thái đã ăn / không ăn
  - Ví dụ:
    - 1 biến bool: đã ăn / không ăn
    - Ở bước sau sẽ thêm đốt và đặt lại biến này
  - Ví dụ:
    - 1 biến int: số quả đã ăn (đề phòng ăn liên tiếp)
    - Ở bước sau nếu biến  $> 0$  thì thêm đốt và giảm biến này đi 1
    - Bài này: dùng cách dưới

# Ăn cherry

```
Position newPosition = head->position.move(direction);  
CellType type = playGround->getCellState(newPosition);
```



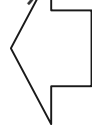
Tạo hàm trong  
PlayGround lấy  
trạng thái ô vuông

```
changePlayGroundState(CELL_EMPTY);  
if (cherry > 0) {  
    cherry--;  
    head = new SnakeNode(newPosition, head);
```



Thêm một đốt nếu  
vừa ăn cherry

```
} else {  
    for (SnakeNode* p = head; p != nullptr; p = p->next)  
        std::swap(p->position, newPosition);  
}
```



Trường hợp không ăn,  
trườn lên phía trước,  
***hãy tìm hiểu xem đoạn  
mã này làm việc thế nào  
?***

```
changePlayGroundState(CELL_SNAKE);
```

```
if (type == CELL_CHERRY) {  
    cherry++;  
    playGround->addCherry();  
}
```



Đánh dấu đã ăn cherry

# Thêm quả cherry sau khi ăn

Cherry mới xuất hiện ngẫu nhiên trong các ô trống (CELL\_EMPTY)

Phiên bản 0.2: rắn ăn quả dài ra

<https://github.com/tqlong/advprogram/archive/200c4c2bc74012548712263e99b78395ad8b6de2.zip>

```
void PlayGround::addCherry()
{
    do {
        Position p(rand()%getWidth(), rand()%getHeight());
        if (getCellState(p) == CELL_EMPTY) {
            changeCellState(p, CELL_CHERRY);
            break;
        }
    } while (true);
}
```

# Phiên bản 0.3: xử lý va chạm

- Các trường hợp thua cuộc
  - Va chạm với cạnh màn hình
    - Sau này có thể ẩn loại quả cho phép đi xuyên qua bên kia màn hình
  - Va chạm với thân rắn
    - Tương tự, có loại quả cho phép đi xuyên qua thân rắn
- Cần kiểm tra xem newPosition có hợp lệ hay không

# Thử lần 1

```
Position newPosition = head->position.move(direction);  
if (!playGround->checkPosition(newPosition)) return;
```

```
bool PlayGround::checkPosition(Position pos)  
{  
    if ( !pos.isInsideBox(0,0,getWidth(),getHeight())  
        || getCellState(pos) == CELL_SNAKE ) {  
        status = GAME_LOST;  
        return false;  
    }  
    return true;  
}
```

# Thử lần 1

- Đã xử lý được va chạm với cạnh
- Xử lý đa phần các trường hợp va chạm với thân rắn
- Trường hợp rắn đủ dài để “cắn đuôi”
  - Chương trình hiện tại sẽ báo thua cuộc và thoát
  - Xử lý thế nào ?
    - Chuyển kiểm tra hợp lệ vào Snake
    - Cho phép newPosition trùng với đuôi rắn
    - Làm hàm setStatus ở Playground



# Thử lần 2

```
Position newPosition = head->position.move(direction);  
if (!checkPosition(newPosition)) {  
    playGround->setGameStatus(GAME_LOST);  
    return;  
}
```

Không tính đốt  
đuôi khi kiểm tra  
hợp lệ

Đốt đuôi là đốt  
có  
`next == nullptr`



```
bool Snake::checkPosition(Position pos)  
{  
    if ( !pos.isInsideBox(0,0,  
        playGround->getWidth(), playGround->getHeight()) )  
        return false;  
  
    for (SnakeNode* p = head; p->next != nullptr; p = p->next)  
        if (p->position == pos)  
            return false;  
  
    return true;  
}
```

# Toán tử so sánh 2 Position

Có thể tự định nghĩa toán tử ==, !=, +, - cho kiểu dữ liệu Position

```
bool Position::operator==(Position p) {  
    return x == p.x && y == p.y;  
}
```

**Phiên bản 0.3: kiểm tra và chạm**

<https://github.com/tqlong/advprogram/archive/b4565b2e0b8caf10be65025f1db67cc94dafbbcb.zip>

# Phiên bản 0.4: vẽ đẹp hơn

- Mục tiêu:
  - Hình rắn sinh động: đầu, đuôi, thân, các khúc cua ...
  - Hình quả đẹp
- Kỹ thuật:
  - Sử dụng `SDL_Texture` và `loadTexture()` của `Painter` để đọc ảnh vẽ sẵn, đẹp từ file `JPG`, `PNG`

# Phiên bản 0.4: vẽ đẹp hơn

- Ở các phiên bản trước để có chương trình nhanh ta chưa cấu trúc code vẽ, cần
  - hàm vẽ đường ngang: `drawHorizontalLine`
  - hàm vẽ đường dọc: `drawVerticalLine`
  - hàm vẽ quả cherry: `drawCherry`
  - hàm vẽ rắn: `drawSnake`
- Các hàm này cần vẽ ở tọa độ tương đối với 1 điểm (left,top) nào đó
  - Để sau này có thể phải di chuyển khung vẽ

# Vẽ đường ngang, dọc

```
void drawVerticalLine(Painter& painter, int left, int top, int cells)
{
    painter.setColor(WHITE_COLOR);
    painter.setAngle(-90);
    painter.setPosition(left, top);
    painter.moveForward(cells * CELL_SIZE);
}
```

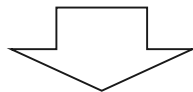
```
void drawHorizontalLine(Painter& painter, int left, int top, int cells)
{
    painter.setColor(WHITE_COLOR);
    painter.setAngle(0);
    painter.setPosition(left, top);
    painter.moveForward(cells * CELL_SIZE);
}
```

# Vẽ cherry, vẽ các ô của rắn

```
void drawCherry(Painter& painter, int left, int top)
```

```
{  
    painter.setColor(ORANGE_COLOR);  
    painter.setAngle(-90);  
    painter.setPosition(left+5, top+5);  
    painter.createSquare(CELL_SIZE-10);  
}
```

Sử dụng vector thay cho danh sách liên kết, bởi nếu truyền con trỏ head thì bên ngoài có thể thay đổi vị trí các đốt của rắn



```
void drawSnake(Painter& painter, int left, int top, vector<Position> positions)
```

```
{  
    painter.setColor(RED_COLOR);  
    painter.setAngle(0);  
    for (Position pos : positions) {  
        painter.setPosition(left+pos.x*CELL_SIZE+5, top+pos.y*CELL_SIZE+CELL_SIZE/2);  
        painter.createCircle(CELL_SIZE/2-5);  
    }  
}
```

# Lấy vị trí các đốt rắn

Dùng hàm **const** trong PlayGround để bảo vệ dữ liệu về rắn

```
vector<Position> PlayGround::getSnakePositions() const
{
    vector<Position> res;
    for (SnakeNode* p = snake.getHead(); p != nullptr; p = p->next)
        res.push_back(p->position);
    return res;
}
```

```
void renderGamePlay(Painter& painter, const Playground& playground)
{
    int top = 0, left = 0;
    int width = playground.getWidth(), height = playground.getHeight();
    painter.clearWithBgColor(PURPLE_COLOR);

    for (int i = 0; i <= width; i++)
        drawVerticalLine(painter, left+i*CELL_SIZE, top+0, height);
    for (int i = 0; i <= height; i++)
        drawHorizontalLine(painter, left+0, top+i * CELL_SIZE, width);

    const vector<vector<CellType> >& squares = playground.getSquares();
    for (int i = 0; i < height; i++)
        for (int j = 0; j < width; j++)
            if (squares[i][j] == CELL_CHERRY)
                drawCherry(painter, left+j*CELL_SIZE, top+i*CELL_SIZE);

    vector<Position> snakePositions = playground.getSnakePositions();
    drawSnake(painter, left, top, snakePositions);
    SDL_RenderPresent(painter.getRenderer());
}
```



# Vẽ quả cherry

- Chọn một ảnh đẹp cho quả cherry
- Ghi vào đĩa thành file **cherry.png**
- Dùng Painter::loadTexture đọc ảnh
  - Đọc ảnh 1 lần lúc chương trình khởi động
- Dùng Painter::createImage vẽ ảnh
  - Sửa hàm createImage để cho phép vẽ ảnh vào 1 hình chữ nhật trong cửa sổ
- Do sẽ có nhiều ảnh trong trò chơi, tạo lớp Gallery để quản lý ảnh



# Lớp Gallery

```
enum PictureID { PIC_CHERRY = 0, };
```

```
class Gallery
```

```
{
```

```
    std::vector<SDL_Texture*> pictures;  
    Painter& painter;
```

```
public:
```

```
    Gallery(Painter& painter_);  
    ~Gallery();
```

```
    void loadGamePictures();
```

```
    SDL_Texture* getImage(PictureID id) const { return pictures[id]; }
```

```
};
```



Danh sách các  
SDL\_Texture  
chứa các ảnh  
theo thứ tự  
PictureID



Đọc các ảnh  
theo thứ tự trên

# Lớp Gallery

```
Gallery::Gallery(Painter& painter_) : painter(painter_)
{
    loadGamePictures();
}
```

```
Gallery::~~Gallery()
{
    for (SDL_Texture* texture : pictures)
        SDL_DestroyTexture(texture);
}
```



Hủy các ảnh đã  
đọc khi hủy đối  
tượng Gallery

```
void Gallery::loadGamePictures()
{
    pictures.push_back(painter.loadTexture("cherry.png"));
}
```

# Sửa hàm Painter::createImage

Thêm khả năng đưa ảnh vào vị trí bất kì trên cửa sổ

```
bool createImage( SDL_Texture* texture,
                  SDL_Rect* srcrect = nullptr, SDL_Rect* dstrect = nullptr );

bool Painter::createImage( SDL_Texture* texture,
                           SDL_Rect* srcrect, SDL_Rect* dstrect)
{
    if( texture == NULL ) return false;
    SDL_RenderCopy( renderer, texture, srcrect, dstrect );
    return true;
}
```

# Sử dụng Gallery vẽ quả cherry

```
Gallery* gallery = nullptr; // global picture manager
```

```
int main(int argc, char* argv[])
{
    ...
    Painter painter(window, renderer);
    gallery = new Gallery(painter);
    ...
    delete gallery;
    quitSDL(window, renderer);
    ...
}
```

```
void drawCherry(Painter& painter, int left, int top)
{
    SDL_Rect dst = { left+5, top+5, CELL_SIZE-10, CELL_SIZE-10 };
    painter.createImage(gallery->getImage(PIC_CHERRY), NULL, &dst);
}
```

Đến đây chương trình đã vẽ được quả cherry đẹp từ file ảnh cherry.png

# Vẽ rắn

- Rắn gồm đốt đầu và các đốt thân
- Khi di chuyển, các đốt thân có thể nằm ngang hoặc dọc
- Vây cần ít nhất 3 ảnh
  - Đầu
  - Thân ngang
  - Thân dọc

# Vẽ rắn

## Thêm ảnh

```
enum PictureID {  
    PIC_CHERRY = 0, PIC_SNAKE_VERTICAL,  
    PIC_SNAKE_HORIZONTAL, PIC_SNAKE_HEAD,  
};
```

```
void Gallery::loadGamePictures()  
{  
    pictures.push_back(painter.loadTexture("cherry.png"));  
    pictures.push_back(painter.loadTexture("snake_vertical.png"));  
    pictures.push_back(painter.loadTexture("snake_horizontal.png"));  
    pictures.push_back(painter.loadTexture("snake_head.jpg"));  
}
```

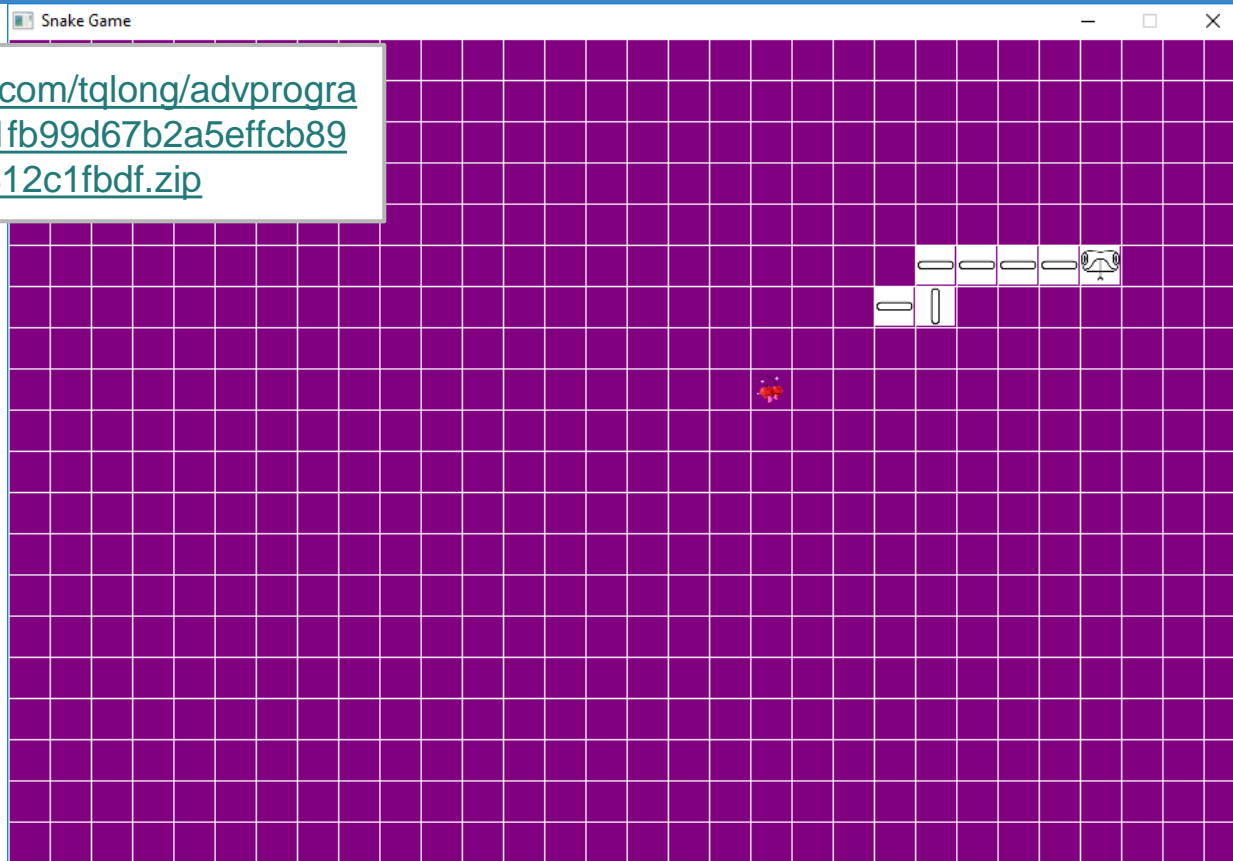
# Vẽ rắn

```
void drawSnake(Painter& painter, int left, int top, vector<Position> pos)
{
    for (size_t i = 0; i < pos.size(); i++) {
        SDL_Rect dst = { left+pos[i].x*CELL_SIZE+1, top+pos[i].y*CELL_SIZE+1,
                        CELL_SIZE-2, CELL_SIZE-2 };
        SDL_Texture* texture = NULL;
        if (i > 0) {
            if (pos[i].y == pos[i-1].y)
                texture = gallery->getImage(PIC_SNAKE_HORIZONTAL);
            else
                texture = gallery->getImage(PIC_SNAKE_VERTICAL);
        } else { // snake's head
            texture = gallery->getImage(PIC_SNAKE_HEAD);
        }
        painter.createImage(texture, NULL, &dst);
    }
}
```



# Phiên bản 0.4

<https://github.com/tqlong/advprogram/archive/691fb99d67b2a5effcb8954141e5a0a812c1fbdf.zip>



# Bài tập

- Thêm ảnh các đốt ở góc
- Thêm loại quả khác
  - Cho phép đi xuyên tường
  - Cho phép đi xuyên rắn
  - Cho phép dài ra nhiều đốt hơn