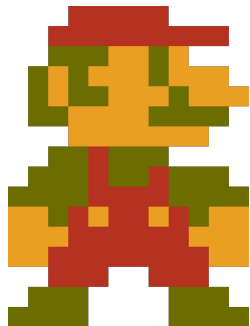




# Pràctica PRO2

## PART 2

Quadrimestre Primavera 2024/25



RogeR Bitlloch | [roger.bitlloch@estudiantat.upc.edu](mailto:roger.bitlloch@estudiantat.upc.edu)

## Part 2: Eficiència

La classe que volem hauria de tenir la següent funcionalitat:

1. Serà un **contenidor**, és a dir, contindrà objectes (del joc). Però només contindrà els punters als objectes, no els objectes en sí. Això és important perquè la classe Game ja té els seus contenidors (ara mateix un `vector<Platform>` per guardar les plataformes), i ja ens va bé. Només volem un altre contenidor que ens resol el problema dels rectangles.

```
#include "platform.hh"
template class Finder<Platform>;

#include "strawberry.hh"
template class Finder<Strawberry>;

template <typename T>
void Finder<T>::add(const T *t) {
    // Afegir al mapa de rectangles
```

Com estem creant un contenidor hem de posar lo del “`template <typename T>`” abans de cada funció i incloure les classes per les quals és un template en el `.cc`

2. Farà la suposició que els objectes que es posin al contenidor **tenen un mètode** `get_rect()`, que retorna un `Rect`. Això és perquè, independentment del tipus d'objecte que tinguem, si aquest objecte té el mètode `get_rect`, el nostre contenidor ja es podrà espabilar i classificar els rectangles d'alguna manera, sense haver de conèixer els altres detalls.

Ja ho hauríeu de tenir de la part 1 però per si de cas ho heu fet d'alguna altre manera, cal implementar `get_rect` pel vostre col·leccionable

```
pro2::Rect get_rect() const {  
    return {left_, top_, right_, bottom_};  
}
```

```
Rect Strawberry::get_rect() const {  
    return {pos_.x-Width/2, pos_.y-Height, pos_.x+Width/2, pos_.y};  
}
```

\*Suposant que la posició es guarda a l'aresta d'abaix al centre del vostre personatge

3. Tindrà 4 mètodes: afegir, esborrar, actualitzar i consultar. Aquests mètodes permetran:

- Afegir un objecte al contenidor.
- Esborrar un objecte del contenidor.
- Actualitzar un objecte que ja estava al contenidor, que vol dir actualitzar el seu rectangle, realment.
- I finalment, el mètode més important: **consultar els objectes de dins el contenidor que estiguin dins d'un cert rectangle**. Aquest mètode el cridarem amb les coordenades de la càmera per saber quins objectes són visibles.

Copieu el .hh del anunciat de pro2

Al ser un contenidor ho podeu fer tot en un sol fitxer .hh en comptes de tenir .cc i .hh

```
/**
 * @brief Retorna el conjunt d'objectes amb rectangles
 *         total o parcialment dins de 'rect'.
 *
 * Si el nombre de rectangles del contenidor és 'n', el
 * cost de l'algorisme ha de ser  $O(\log n)$ .
 *
 * @param rect El rectangle de cerca
 *
 * @returns Un conjunt de punters a objectes que tenen un
 *         rectangle parcial o totalment dins de 'rect'
 */
std::set<const T*> query(pro2::Rect rect) const;
};
```

# Anem a fer-ho d'ineficient $O(n)$ a eficient

Utilitzarem una llista i recorrarem la llista en busca dels rectangles que ens interessin per retornar únicament aquells.

(així comprovem que tot vagi bé i després ja ens barallem per millorar l'eficiència)

```
template <typename T>
void Finder<T>::add(const T *t) {
    objects_.insert(t);
}

template <typename T>
void Finder<T>::remove(const T *t) {
    objects_.erase(t);
}

template <typename T>
void Finder<T>::update(const T *t) {
    remove(t);
    add(t);
}
```

```
template <typename T>
set<const T*> Finder<T>::query(Rect rect) const {
    set<const T*> result;
    typename set<const T*>::const_iterator it;
    for (it = objects_.begin(); it != objects_.end(); ++it) {
        if (rects_overlap((*it)->get_rect(), rect)) {
            result.insert(*it);
        }
    }
    return result;
}
```

# A game.hh i game.cc:

## Declarem a Game.hh

```
Finder<Platform> platform_finder_;  
Finder<Strawberry> strawberry_finder_;  
set<Platform*> visible_platforms_;  
set<Strawberry*> visible_strawberrys_;
```

A game.cc omplim els finders  
amb totes les plataformes i  
monedes

```
// Añadir todas las plataformas al finder  
for (const Platform& platform : platforms_) {  
    platform_finder_.add(&platform);  
}  
  
// Añadir todas las fresas al finder  
for (const Strawberry& strawberry : strawberrys_) {  
    strawberry_finder_.add(&strawberry);  
}
```

# Al fer l'update actualitzem els sets de plataformes i monedes visibles

```
void Game::update(pro2::Window& window) {  
    // Creem un rectangle amb l'area visible actual  
    Pt camera_center = window.camera_center();  
    Rect visible_area = {  
        camera_center.x - window.width() / 2, // left  
        camera_center.y - window.height() / 2, // top  
        camera_center.x + window.width() / 2, // right  
        camera_center.y + window.height() / 2 // bottom  
    };  
    // Actualitzem les plataformes visibles dins el rectangle  
    visible_platforms_ = platform_finder_.query(visible_area);  
    // Actualitzem les maduixes visibles dins el rectangle  
    visible_strawberrys_ = strawberry_finder_.query(visible_area);  
  
    process_keys(window);  
    update_objects(window);  
    update_camera(window);  
}
```

Calculem el rectangle de la càmera i fem `finder.query(rect);` per obtenir aquells elements que apareixen a la càmera



# Quan fem paint o update\_objects, només ho fem en aquells elements visibles, els altres no calen

```
void Game::paint(pro2::Window& window) {  
    window.clear(sky_blue);  
  
    for (Platform* p : visible_platforms_) {  
        p->paint(window);  
    }  
  
    for (Strawberry* s : visible_strawberrys_) {  
        s->paint(window);  
    }  
  
    mario_.paint(window);  
}
```

Pot ser, que et salti error alguna cosa perquè els elements del finder són constants, n'hi ha prou en treure el canviar el query del finder perquè retorni un `set<T>` en comptes d'un `set<const>`

```
void Game::update_objects(pro2::Window& window) {  
    mario_.update(window, platforms_);  
  
    for (Strawberry* s : visible_strawberrys_) {  
        s->update(window);  
  
        if (!s->is_collected() && rects_overlap(mario_.get_rect(), s->get_rect())) {  
            s->collect();  
            ++strawberry_count_;  
            cout << "Maduixa recollida! Tens: " << strawberry_count_ << " maduixes!!!" << endl;  
        }  
    }  
}
```



Prova de posar 10.000 plataformes i **que funcioni correctament i sense lag abans de continuar**

```
for (int i = 1; i < 100000; i++) {  
    platforms_.push_back(Platform(250 + i * 200, 400 + i * 200, 150, 161));  
    strawberries_.push_back(Strawberry({325 + i * 200, 150}));  
}
```

Si tens dubtes d'alguna cosa o et falta alguna part de codi aquí tens penjada la meva implementació ineficient (la que has vist fins ara):

Codi: [PART 2.0 O\(n\)](#)

# Ara anem a millorar l'eficiència de l'estructura de dades finder

- Volem evitar recorre tots els elements que tenim
- Podem fer una **cerca (temps  $\log N$ )** per començar a iterar en els elements que la seva part dreta  $\geq$  esq\_pantalla i acabar d'iterar en aquells que la seva part esquerra  $\geq$  dreta\_pantalla?
- Quines estructura de dades podem utilitzar per fer aquesta cerca?

Podem utilitzar un map, on tinguem els punts on tinguem l'esquerra dels quadrats ordenats

```
private:
    // Mapa que donat un objecte retorna el seu rect
    map<const T*, pro2::Rect> obj_rect_map_;

    // Mapa que donat un left retorna els objectes que comencen allà
    map<int, set<const T*>> x_start_map_;
```

D'aquesta manera, al estar ordenat, podem fer una cerca binària per saber a on hem de començar a buscar

```
template <typename T>
set<const T*> Finder<T>::query(Rect rect) const {
    set<const T*> result;

    // Busco el primer objecte que left <= rect.right
    // Esta operaci3n es O(log n)
    auto it = x_start_map_.lower_bound(rect.left);

    // Itero només amb aquells objectes que left > rect.right
    while (it != x_start_map_.end() && it->first <= rect.right) {
        // Per cada objecte
        for (const T* obj : it->second) {
            // Agafo el rectangle
            const Rect& obj_rect = obj_rect_map_.at(obj);

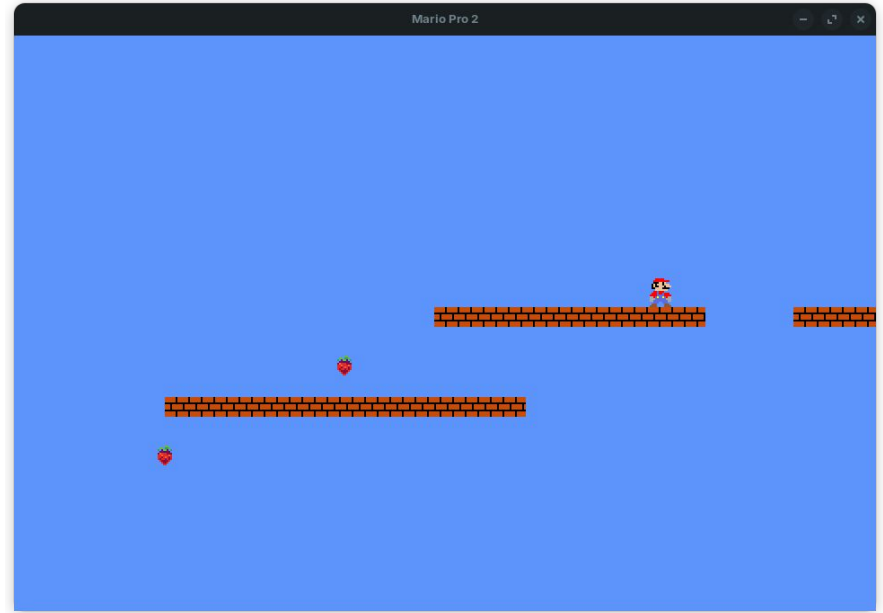
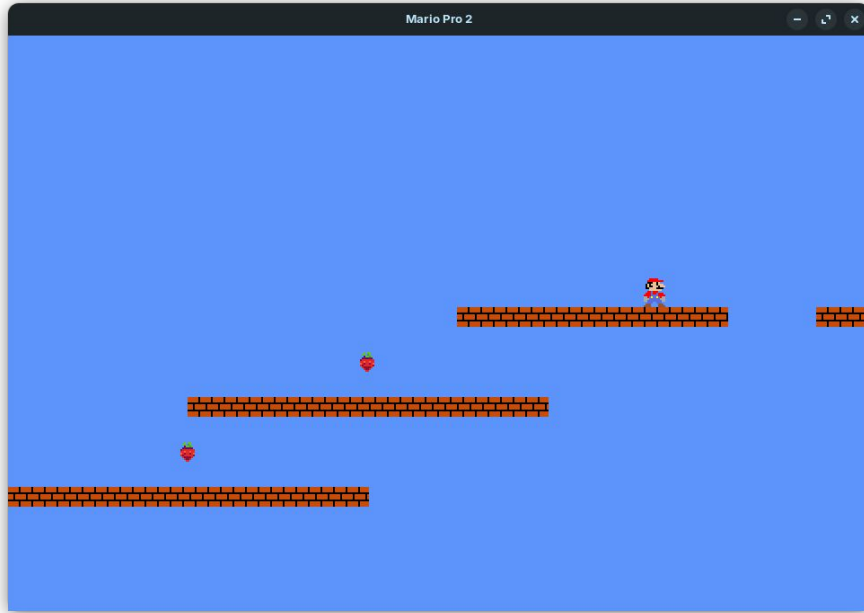
            // Comprovo si esta dins (podria estar abaix o dalt)
            if (obj_rect.left <= rect.right &&
                obj_rect.right >= rect.left &&
                obj_rect.top <= rect.bottom &&
                obj_rect.bottom >= rect.top) {

                result.insert(obj);
            }
        }
        ++it;
    }

    return result;
}
```

# Però tenim un problema...

Com que només comprovo la paret del rectangle de l'esquerra a la que aquesta surt de la pantalla és deixa de pintar, tot i que part de l'objecte encara hi sigui



Codi: [PART 2.1  \$O\(\log n\)\$  SENSE FUNCIONAR](#)

# Podem concloure que...

Llavors necessitem fer una **cerca** al primer objecte que compleixi que té, almenys una aresta del rectangle dins la pantalla

- Per tant necessitem una estructura de dades on poder fer una cerca més “personalitzada” o bé tenir en compte la part dreta a més a més de l'esquerra
- **Deures:** Buscar i provar d'implementar una estructura de dades on pugueu fer la cerca de forma eficient