Universidad de Costa Rica
Facultad de Ingeniería
Escuela de Ciencias de la Computación e Informática

CI-0117: Programación Paralela y Concurrente

Tarea Programada 2

Profesor : José Andrés Mena Arias

Estudiantes :

B82957 : Marco Ferraro Rodríguez
B86524 : Gabriel Revillat Zeledón

Índice

1. Enunciado

En este proyecto se implementará un battle royale entre procesos, simulando una versión muy simplificada del juego Super Mario Bross 35. Para ello se creará un programa en C++ o en C donde un jugador virtual simulará el recorrido de Mario a lo largo del primer mundo (1-1) de su versión original. Dicho programa deberá poder ejecutarse utilizando el estándar de Message Passing Interface (MPI) de forma que se puedan tener varios procesos simultáneamente simulando el juego e interactuando entre ellos.

Para efectos de este proyecto no se va a implementar el juego como tal, sino una simulación entre dos jugadores virtuales. De igual manera no todos los detalles del sistema de batallas estarán dentro del alcance de la simulación, únicamente las que se especifican en los requerimientos de las siguientes secciones.

2. Diseño de la Solución

Para implementar una solución a este problema se plantea usar un Modelo Vista Controlador. En la sección de **Controllers** se plantea tener 8 archivos **.cpp**.

- **main.cpp**

  En este caso, el archivo main se encargara de la sincronización y envio de mensajes usando metodos y funciones de la librería mpi.h.

- **mario.cpp**

  Al utilizar c++, mario.cpp va a ser una clase que su función va a radicar a guardar atributos, en este caso **la estrategia, si esta vivo y la cantidad de monedas que posee.**

- **world.cpp**

  Se planea que el archivo world.cpp mantenga la información del mapa, mediante una **lista de listas**. Este "mapa"va a poseer los elementos y va a encargarse de inicializar el "mapaz tener métodos para que mario itere sobre él.

- **Los Elementos**

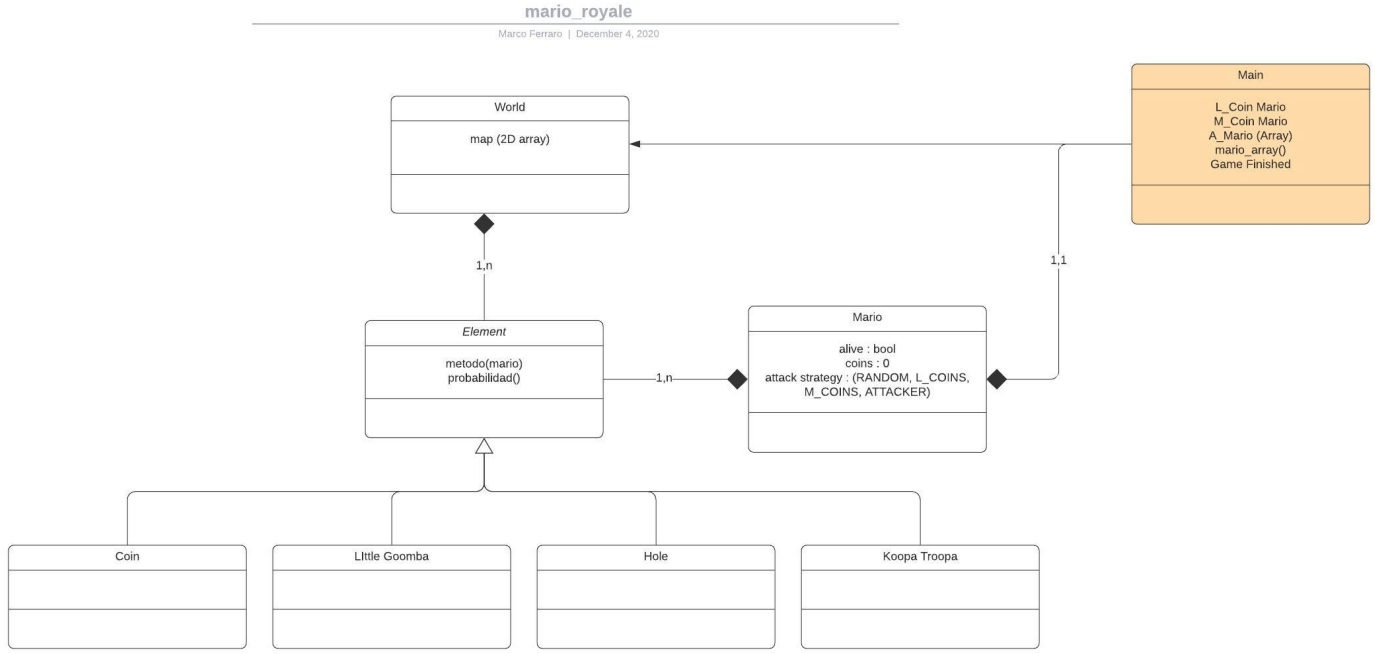
  Para los elementos se planea utilizar polimorfismo, esto con el motivo de tratar de reducir código duplicado. se planea utilizar una clase abstracta definida en la carpeta de modelos; **element.h**. Cada elemento tendrá su método polimórfico que, dependiendo de su tipo, hará acciones diferentes sobre mario.

  - coin.cpp
  - koopa.cpp
  - goomba.cpp
  - hole.cpp

Todas las firmas de los métodos y especificación de estas estructuras están bajo la carpeta de **Models** bajo sus **.h** respectivos.

2.1. Diagrama UML

A continuación se presenta un diagrama para representar los requerimientos de datos para la implementación.



3. Implementación de Solución

A continuación se presentara las definiciones de clases más importantes:

3.1. element.h

```
class Element {
    public:
        virtual int action(Mario &, int probability) = 0;

        // Operators
        bool operator==(const Element &other) {
            return this->element_type == other.element_type;
        };

        bool operator!=(const Element &other) {
            return this->element_type != other.element_type;
        };


    protected:
        std::string element_type;
};      You, 2 weeks ago • Empezando.h
```

Como se explico en la seccion anterior, se tenia la intención de crear una clase abstracta para tener un grupo de clases que puedan heredar de esta. **element.h** consta de la definición de un método que recibe a mario de parámetro, además de unos operadores para que los elementos puedan compararse entre si, con el motivo para hacer comparación de elementos mas fácil.

```
Coin::Coin() {
    this->element_type = COIN;
}

int Coin::action(Mario &mario, int probability) {
    int status = ELEMENT_IGNORED_BY_MARIO;

    if (probability > 50) {
        // falta el print
        mario.add_coin(1);
        status = ELEMENT_KILLED_BY_MARIO;
    }

    return status;
}
```

```
Little_Goomba::Little_Goomba() {
    this->element_type = LITTLE_GOOMBA;
}



int Little_Goomba::action(Mario &mario, int probability) {
    int status = ELEMENT_IGNORED_BY_MARIO;

    if (probability > 0 && probability <= 5) {
        mario.set_inactive();
        status = ELEMENT_KILLED_MARIO;
    } else {
        if (probability > 5 && probability <= 60) {
            // mario salta y pasa
            // no tiene efecto
        } else {
            if (probability > 60 && probability <= 100) {
                // mario salta y mata al enemigo
                status = ELEMENT_KILLED_BY_MARIO;
            }
        }
    }
    return status;
}       You, a week ago • Working
```

Como se puede ver en las imagenes, cada tipo de elemento va a actuar de manera distinta dependiendo de la probabilidad obtenida.

3.2. mario.h

```
gabrielrevillat, a day ago | 2 authors (You and others)
class Mario {
    public:
        Mario(int);
        Mario(char);
        void add_coin(int);
        int get_coins_amount();
        bool is_active();
        void set_inactive();
        std::string get_attack_strategy();
        void set_attack_strategy(char);

    private:
        std::string attack_strategy;
        // amount of coins
        int coins;
        // bool to know if mario is active or not
        bool active;
        // mpi rank, needs to be implemented
        int my_id;
};
```

mario.h estará funcionando casi como una estructura de datos, los únicos valores cambiantes van a ser la cantidad de monedas y si esta inactivo o no. Cabe destacar que cada mario va a tener su estrategia de batalla, la cual va a ser guardada como un atributo de la clase.

3.3. world.h

```
gabrielrevillat, 21 hours ago | 3 authors (Gabriel and others)
class World {
    public:
        World();
        //~World();

        void initialize_world();
        std::vector<Element*> get_next_position_elements(int position);
        void add_goomba(int world_position);
        void add_koopa(int world_position);
        void remove_coin(int world_position);
        void remove_goomba(int world_position);
        void remove_koopa(int world_position);
        void print_world_array();

    private:
        std::vector< std::vector<Element *> > world_array;
};
```

Como se puede notar, world tiene definidos métodos para agregar o borrar elementos del mapa. El mapa esta definido mediante una estructura de datos tipo vector de vectores implementada mediante librerías externas de c++.

4. Funciones MPI

A continuación se mencionan el grupo de herramientas de MPI que están en el programa y como se utilizan. Cabe reiterar que estas funciones están implementadas en **main.cpp**

- **MPI_Bcast**

  Solo se esta utilizando MPI_Bcast a lo largo del programa, el motivo es que todos los procesos estén transmitiendo el id del proceso que el usuario desea ver.

- **MPI_Allreduce**

  Se implementó MPI_Allreduce para utilizarlo con 2 motivos diferentes. El primero es para tener un contador de todos los marios que se encuentran activos, y el segundo es para conocer la **cantidad** de marios que poseen la estrategia **attacker** que siguen activos.

- **MPI_Allgather**

  Esta función de MPI es la que más se usa a lo largo del programa. Recordemos que MPI_Allgather junta datos para armar un arreglo de datos, valga la redundancia. Se utiliza el Allgather sobre una serie de arreglos que tiene la finalidad de mantener control y conocer el estado de cada proceso.

  - coins_array: Este arreglo es para conocer la cantidad de monedas de cada proceso en cada momento.
  - active_marios: Este arreglo nos ayuda a conocer si un mario esta activo o no, para poder realizar el ataque.
  - attacking_array: Este arreglo es para conocer que proceso esta atacando cual. Recordemos, que por la funcionalidad del programa, un proceso puede que no este siendo atacado, pero siempre va a buscar atacar a otro.
  - attackers: Este arreglo tiene la finalidad de conocer **cuales son los procesos que tienen la estrategia de attacker**.
  - goombas y koopas: Se vio la necesidad de implementar estos arreglos con la finalidad de cuantos koopas o goombas tiene que ïngresar a su mundoün proceso.

5. Manual de usuario

Para compilar y correr el proyecto se necesita un compilador de mpi, en este caso se esta probando con el compilador mpicc.

5.0.1. Compilación

En el proyecto se adjunta un archivo **MakeFile**. Para la compilación basta escribir desde la terminal **make** siempre cuando este corriendo un S.O. Linux.

```
mantofer@marco-hp-envy:~/Desktop/CI0117-2020-S2/Proyectos/super_mario_mpi$ mpiexec -n 3 ./super_mario_mpi 2 L
World pos. 0: Mario #2 is walking. Coins: 0 | attacking #0 | being attacked by #0 | attack strategy: less_coins | Total playing: 2
World pos. 1: Mario #2 is walking. Coins: 0 | attacking #0 | being attacked by #0 | attack strategy: less_coins | Total playing: 2
World pos. 2: Mario #2 is walking. Coins: 0 | attacking #0 | being attacked by #0 | attack strategy: less_coins | Total playing: 2
World pos. 3: Mario #2 is walking. Coins: 0 | attacking #0 | being attacked by #0 | attack strategy: less_coins | Total playing: 2
World pos. 4: Mario #2 is walking. Coins: 0 | attacking #0 | being attacked by #0 | attack strategy: less_coins | Total playing: 2
World pos. 5: Mario #2 is walking. Coins: 0 | attacking #0 | being attacked by #0 | attack strategy: less_coins | Total playing: 2
World pos. 6: Mario #2 is walking. Coins: 0 | attacking #0 | being attacked by #0 | attack strategy: less_coins | Total playing: 2
World pos. 7: Mario #2 didn't jump and ingored the coin! Coins: 0 | attacking #0 | being attacked by #0 | attack strategy: less_coins
World pos. 8: Mario #2 is walking. Coins: 0 | attacking #0 | being attacked by #0 | attack strategy: less_coins | Total playing: 2
World pos. 9: Mario #2 is walking. Coins: 0 | attacking #0 | being attacked by #0 | attack strategy: less_coins | Total playing: 2
```

Para ejecutar el programa, se escribe en la termina estos comandos **mpiexec -n *cantidad de procesos* ./super_mario_mpi *el proceso que desea ver la sigla de la estrategia deseada***

Cabe destacar, que por requerimientos estamos ignorando el proceso 0, por lo tanto, si se desea utilizar 3 procesos solo se podrá ver la información del proceso 1 o 2.

```
World pos. 28: Mario #2 is walking. Coins: 3 | attacking #1 | being attacked by #1 | attack strategy: random | Total playing: 2
World pos. 29: Mario #2 didn't jump and was killed by a little goomba! Coins: 3 | attacking #1 | being attacked by #1 | attack strategy: random | To
World pos. 29: Mario #2 Game Over.
He dead. Enter the number of another player: 1
World pos. 31: Mario #1 is walking. Coins: 2 | attacking #0 | being attacked by #0 | attack strategy: random | Total playing: 1
World pos. 32: Mario #1 is walking. Coins: 2 | attacking #0 | being attacked by #0 | attack strategy: random | Total playing: 1
World pos. 33: Mario #1 jumped and passed the hole! Coins: 2 | attacking #0 | being attacked by #0 | attack strategy: random | Total playing: 1
World pos. 34: Mario #1 jumped and passed the hole! Coins: 2 | attacking #0 | being attacked by #0 | attack strategy: random | Total playing: 1
```

Además, cuando el mario de un proceso se declara inactivo, el usuario podrá ingresar el id de otro proceso para ver la información de este.

6. Conclusiones

A medida de trabajo de grupo, se trabajo mediando github con ramas distintas, bajo el mismo repositorio.

Si se encontraron una serie de anomalías, como por ejemplo la creación de números aleatorios y el linking del compilador, que se tuvo que modificar el diseño original del proyecto para arreglarlo