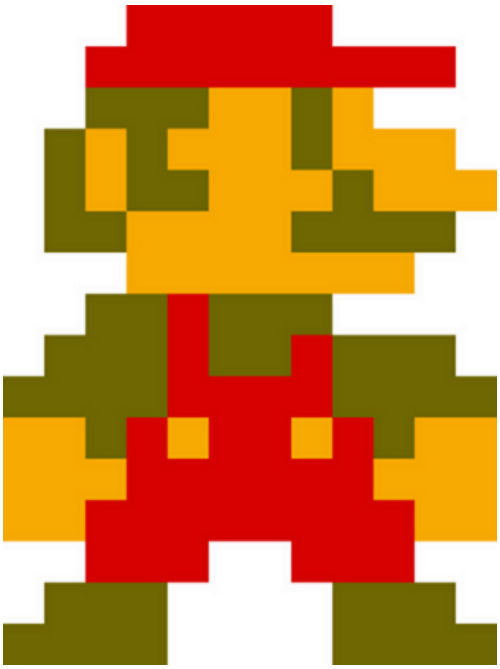
Universidad de Costa Rica

Escuela de Ciencias de la Computación e informática

CI0117- Programación Paralela y Concurrente

Estructura y Funcionalidades del programa:

“Simulador de Battle Royale de Procesos [super_mario_mpi]”



Estudiantes:

Gianfranco Bagnarello Hernández B70866

Katherine González Arias B22867

Profesor: Jose Andrés Mena Arias

II Semestre, diciembre 2020

Para la programación del proyecto se utilizaron varias clases siguiente la estrategia de divide y vencerás, donde se crearon dos folders llamados Models y Controllers.

En el folder de Models se encuentran las clases de entities.cpp, mario.cpp y world.cpp.

entities.cpp: Me crea una lista de entidades que serán ubicadas en la clase world.cpp, además posee los métodos para poder agregar, obtener y eliminar entidades del mundo.

mario.cpp: La clase Mario es la más amplia en el sentido de lógica. En ella encontramos el id de Mario, su ubicación en el mundo, la cantidad de monedas que posee, si se encuentra activo, la estrategia que utilizará al iniciar un partida para atacar a sus contrincantes de los demás procesos y su respectivo mundo. Con estos atributos podemos encontrar métodos para obtener la ubicación en el mundo, obtener la cantidad de monedas que posee, agregar monedas, devolver si está activo o no.

Con la creación del método generateRandomInt(), se puede crear métodos donde la vamos a utilizar para que Mario elija la estrategia inicial de ataque a los demás procesos ya sea R que representa random, L para acatar a quien tiene menos mones, M para atacar a quien tiene más monedas y A para que un jugador A ataque a uno de los jugadores atacantes cada vez que elimina un enemigo.

El método de getWorldBlock(int position) nos permite obtener la ubicación de la entidades en el mundo.

Para manejar la manera en la que Mario va a reaccionar a las entidades que se va encontrando en el mundo, ya sean enemigos del tipo little goomba o koopa tropa, obstáculos del tipo hole, además de compensaciones del tipo coin, se utiliza un multimapa getEncounterOutcomes(list blockEntities) y setEncounterResults(multimap outcomes), que nos va a mapear las decisiones que Mario toma al encontrarse con las entidades y nos va a ir imprimiendo en consola las decisiones que fueron tomadas por Mario. Esta decisiones fueron tomadas de acuerdo con probabilidades programados en el método de calculateProbabilityResult(int probability) y calculateEncounterResult(vector probs), que , seguidamente, de acuerdo con los métodos de getHoleEncounterOutcome(), getCoinEncounterOutcome(), getLittleGoombaEncounterOutcome() y getKoopaTroopaEncounterOutcome()se van a ir generando.

world.cpp: En la clase World se crea un arreglo de entidades world de 100 posiciones que nos va a ir ubicando en una locación las diferentes entidades que Mario se puede ir encontrando en la simulación, donde podemos encontrar casos que en una misma ubicación del mundo tenemos varias entidades diferentes e inclusive puede haber entidades del mismo tipo en una locación del mundo.

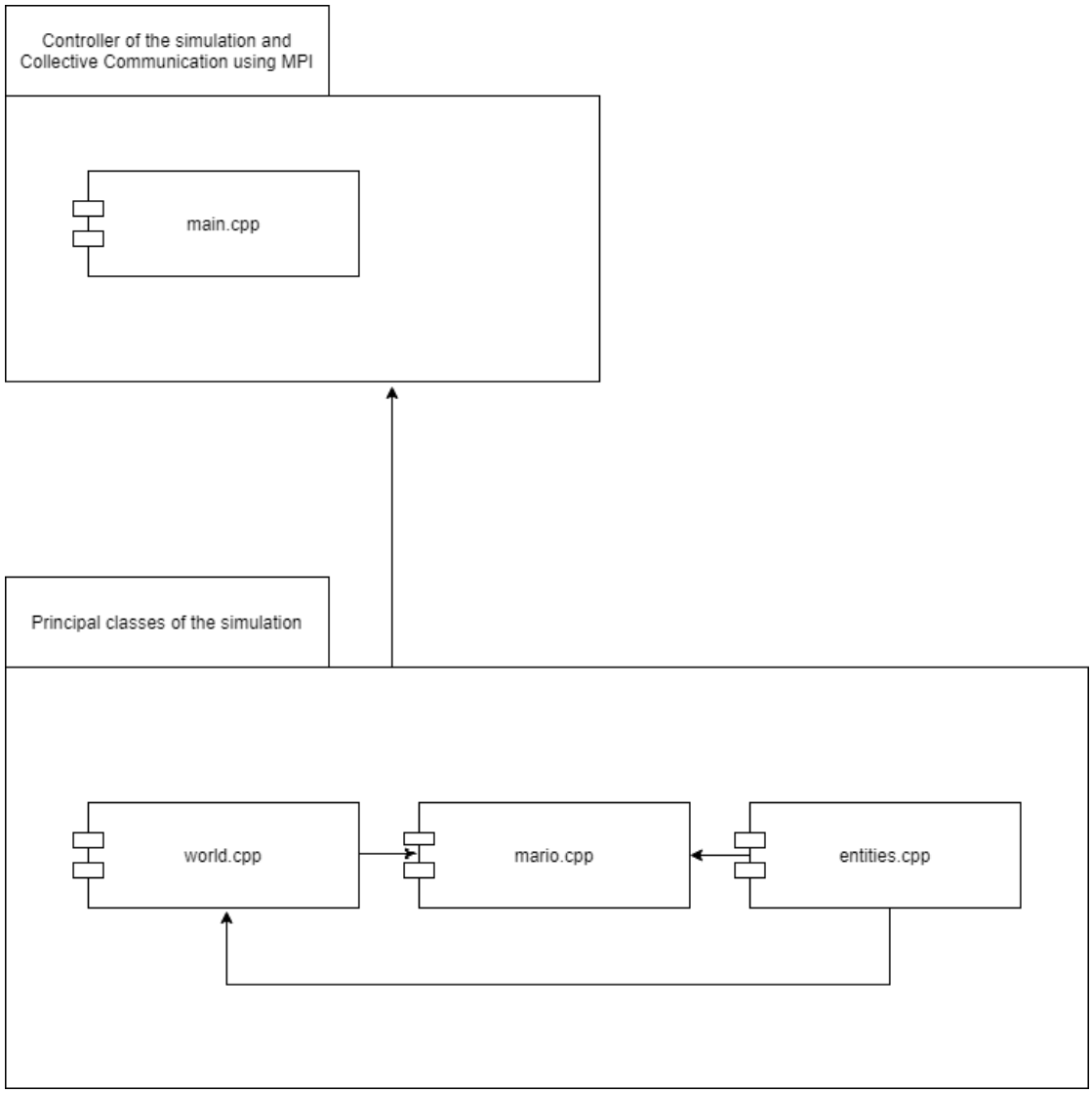
Además, esta clase contiene métodos que nos permite obtener un bloque o ubicación específica del arreglo del mundo o remover una entidad de algún bloque de ubicación e imprimir el mundo.

Ahora bien, en el folder de Controlles se encuentra el Makefile y la clase principal o Main.

En el **Makefile** podemos encontrar las instrucciones que el compilador interpreta para compilar el programa de forma óptima.

La clase **main.cpp** nos permite validar las entradas de los argumentos en consola, nos ayuda a ubicar el número de procesos Mario que van a estar ejecutándose paralelamente en el programa. También la clase main es la que nos permite usar las directivas de MPI para implementar la comunicación colectiva entre procesos y que se vaya ejecutando de forma efectiva la paralelización de procesos.

A continuación, se presentará un diagrama de clases utilizadas en este programa.



Donde la entidad Mario y World reciben funciones de la clase Entities, a la vez que al clase Mario recibe a World y se va hacia la clase main donde es ejecutado el programa.

Funcionalidades de MPI empleadas en este proyecto en el método main.

Para lograr la comunicación colectiva entre los procesos, se utiliza MPI_Reduce para cada Mario o proceso, donde van a tener su propia variable local que contiene el mínimo y el máximo de monedas, entonces con MPI_MIN y MPI_MAX, se va a conseguir cual proceso es el que posee la menor o la mayor cantidad de monedas y con esto se consigue el id del proceso y si es el Mario elegido el main imprime o no imprime.

Limitaciones:

Dentro de la clase Mario se encuentran las funciones para recibir y enviar ataques entre jugadores (procesos), por motivos de tiempo no se logró implementar el uso de MPI_Send y MPI_Recieve para usar estas funciones y lograr que los jugadores se comunicaran ataques. El programa corre de manera concurrente el Mario elegido por el usuario y los controlados por la computadora, cada Mario conoce la cantidad de jugadores que quedan activos, pero no se comunican ataques entre ellos.

Cada proceso tiene un registro del ID del jugador/proceso que lleva más monedas hasta el momento, esto se logra con MPI_Reduce (dos veces, se usa dos veces, una para calcular el que tiene menos y otra el que tiene más). Cada proceso además escoge a quién atacar con base en esto (y otros factores), además prepara el ataque para enviarlo, pero nunca los envía.