



#AGE RACER DESAFÍO FINAL

2021-2022

AGE - P.ISASI & Y. SAEZ

1

INTRODUCCIÓN

- El desafío final consiste en desarrollar un controlador de bots
- Estos bots deben participar en un juego
- Estas son las reglas del juego (cont.)

REGLAS DEL JUEGO



- Este es un juego de optimización en el que debes programar una vaina espacial para conducir a través de una serie de puntos de control lo más rápido posible
- Cada punto de control (checkpoint) está ubicado en la posición indicada con x,y.
- El “checkpointIndex” indica el orden de los puntos de control dados como entrada inicial
- El juego tiene lugar en un mapa de 16.000 unidades de ancho y 9.000 unidades de alto. La coordenada X = 0, Y = 0 es el píxel superior izquierdo

REGLAS DEL JUEGO

- Los puestos de control funcionan de la siguiente manera:
 - Los puestos de control son circulares, con un radio de 600 unidades.
 - La disposición de los puntos de control la establecen los casos de prueba (ficheros `test_case` en formato JSON).
 - No hay puntos de control superpuestos
- El juego acaba tras 600 turnos o cuando se hayan superado todos los puntos de control durante 3 vueltas (lo que ocurra antes)
- El punto de control (checkpoint) se dará por completado si el centro de la vaina espacial entra dentro del círculo de 600 unidades
- Para conducir la vaina espacial se debe especificar las coordenadas de destino X e Y y un grado de aceleración
- El coche irá hacia el destino con un radio de giro máximo de 18 grados y tras el giro aplicará la aceleración
- El juego termina si se envía una orden incorrecta o imposible
- Si el jugador no realiza un movimiento correcto en el tiempo adecuado es eliminado inmediatamente (1.000ms para el primer movimiento y 50ms para los turnos)



REGLAS DEL JUEGO



- En cada turno, los movimientos se calculan de la siguiente manera:
 - **Aceleración:** el coche gira hacia su destino, en un máximo de 18°. El vector de rumbo normalizado del automóvil se multiplica por la potencia de empuje (aceleración) dividida por la masa. El resultado se agrega al vector de velocidad actual
 - **Movimiento:** su vector de velocidad se agrega a la posición de todos los objetos para calcular sus nuevas posiciones.
 - **Fricción:** el vector de velocidad se multiplica por una constante, luego se trunca. La constante es:
 - 0,85 para vainas espaciales,
 - Los valores de posición se truncan
 - El ángulo de rumbo de los coches se expresa en grados y se redondea
 - Los ángulos se proporcionan en grados y en relación con el eje X (Este = 0 grados, Sur = 90 grados).

DATOS PARA EL JUEGO – EN CADA TURNO



- El programa primero debe leer los datos de inicialización de la entrada estándar
- Primera línea: un número entero de puntos de control, la cantidad de todos los puntos de control por pasar (todos los puntos de control se repiten 3 veces)
- Próximas líneas de puntos de control: una línea por punto de control
- Cada punto de control está representado por 2 números enteros: punto de controlX, punto de controlY.
- Datos que nos proporciona el juego:
- Las siguientes líneas son una por entidad con este formato (6 enteros)
 - `checkpointIndex, x, y, vx, vy, angle`
 - `checkpointIndex` indica el índice del siguiente punto de control como se indica en las entradas iniciales. `x, y` para la posición de la entidad.
 - `vx, vy` para el vector de velocidad de la entidad.
 - `angle`. Ángulo de rumbo en grados entre 0 y 360 para el automóvil.
- Entrada de datos para el bot:
 - Una línea para su automóvil: tres enteros X, Y y aceleración
 - Se puede añadir un texto al final que se mostrará encima del automóvil para poder trazar qué estrategia está siguiendo (recomendable)

DATOS DE ENTRADA – EN CADA TURNO

- Una línea para conducir la vaina espacial
- Cada línea de entrada son 3 enteros, X, Y y thrust, opcionalmente se puede añadir una cadena de texto informativa
- Restricciones:

$9 \leq checkpoints \leq 24$

Puntos de control

$0 \leq thrust \leq 200$

Aceleración

$0 \leq angle \leq 360$

Ángulo

Tiempo de respuesta para el primer turno ≤ 1000 ms

Tiempo de respuesta por turno ≤ 50 ms



EJEMPLO EN PYTHON

```
import sys
import math

checkpoints = int(input()) # Count of checkpoints to read
for i in range(checkpoints):
    # checkpoint_x: Position X
    # checkpoint_y: Position Y
    checkpoint_x, checkpoint_y = [int(j) for j in input().split()]

    # game loop
    while True:
        # checkpoint_index: Index of the checkpoint to lookup in the checkpoints input, initially 0
        # x: Position X
        # y: Position Y
        # vx: horizontal speed. Positive is right
        # vy: vertical speed. Positive is downwards
        # angle: facing angle of this car
        checkpoint_index, x, y, vx, vy, angle = [int(i) for i in input().split()]

        # Write an action using print
        # To debug: print("Debug messages...", file=sys.stderr, flush=True)

        # X Y THRUST MESSAGE
        print("5000 5000 200 message")
```

EJEMPLO EN JAVA

```
import java.util.*;
import java.io.*;
import java.math.*;

class Player {

    public static void main(String args[]) {
        Scanner in = new Scanner(System.in);
        int checkpoints = in.nextInt(); // Count of checkpoints to read
        for (int i = 0; i < checkpoints; i++) {
            int checkpointX = in.nextInt(); // Position X
            int checkpointY = in.nextInt(); // Position Y
        }

        // game loop
        while (true) {
            int checkpointIndex = in.nextInt(); // Index of the checkpoint to lookup in the checkpoints input, initially 0
            int x = in.nextInt(); // Position X
            int y = in.nextInt(); // Position Y
            int vx = in.nextInt(); // horizontal speed. Positive is right
            int vy = in.nextInt(); // vertical speed. Positive is downwards
            int angle = in.nextInt(); // facing angle of this car

            // Write an action using System.out.println()
            // To debug: System.err.println("Debug messages...");

            System.out.println("5000 5000 200 message"); // X Y THRUST MESSAGE
        }
    }
}
```

EJEMPLO EN JAVA

Es posible añadir cualquier código sobre la clase del agente para poder mejorar su comportamiento

```
import java.util.ArrayList;
import java.util.Scanner;

public class Agent2 {
    // This agent slows down from 200 to 50 when is at 4000 units before reaching the checkpoint
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        int checkpoints = Integer.parseInt(scanner.nextLine());
        ArrayList<Point> targets = new ArrayList<>();
        for(int i = 0; i < checkpoints; i++){
            String[] line = scanner.nextLine().split( regex: " " );
            System.err.println(line[0] + " " + line[1]);
            targets.add(new Point(Integer.parseInt(line[0]), Integer.parseInt(line[1])));
        }
        double dist = 100000.0;
        int z = 0;
        while (true) {
            String s = scanner.nextLine();
            System.err.println(s);
            String[] input = s.split( regex: " " );
            // id x y vx vy angle
            int target = Integer.parseInt(input[0]);
            int x = Integer.parseInt(input[1]);
            int y = Integer.parseInt(input[2]);
            int vx = Integer.parseInt(input[3]);
            int vy = Integer.parseInt(input[4]);
            Point targ = targets.get(target);

            Point current = new Point(x, y);
            int thrust = 200;
            if(targ.distance(current) < 4000){
                thrust = 50;
            }
            // if(vx < 0)
            //     System.out.println("EXPERT 18 200");
            //else
            System.out.println(10000+ " " + 4000 + " " + thrust + " Agent 2"); // X Y THRUST MESSAGE
        }
    }

    private static class Point{
        public int x, y;
        public Point(int x, int y){
            this.x = x;
            this.y = y;
        }

        double distance(Point p) {
            return Math.sqrt((this.x - p.x) * (this.x - p.x) + (this.y - p.y) * (this.y - p.y));
        }
    }
}
```



SIMULADOR

- **SkeletonMain.java**

```
import com.codingame.gameengine.runner.SoloGameRunner;

public class SkeletonMain {
    public static void main(String[] args) {
        // Uncomment this section and comment the other one to create a Solo Game
        /* Solo Game */
        SoloGameRunner gameRunner = new SoloGameRunner();
        // Sets the player
        gameRunner.setAgent(Agent1.class);
        // Sets a test case
        gameRunner.setTestCase("test0.json");

        // Another way to add a player for python
        // gameRunner.addAgent("python3 /home/user/player.py");

        // Start the game server
        gameRunner.start();
        // Simulate
        //gameRunner.simulate();
    }
}
```

<http://localhost:8888/test.html>

Mapa / caso de prueba

EJECUTAR SERVIDOR



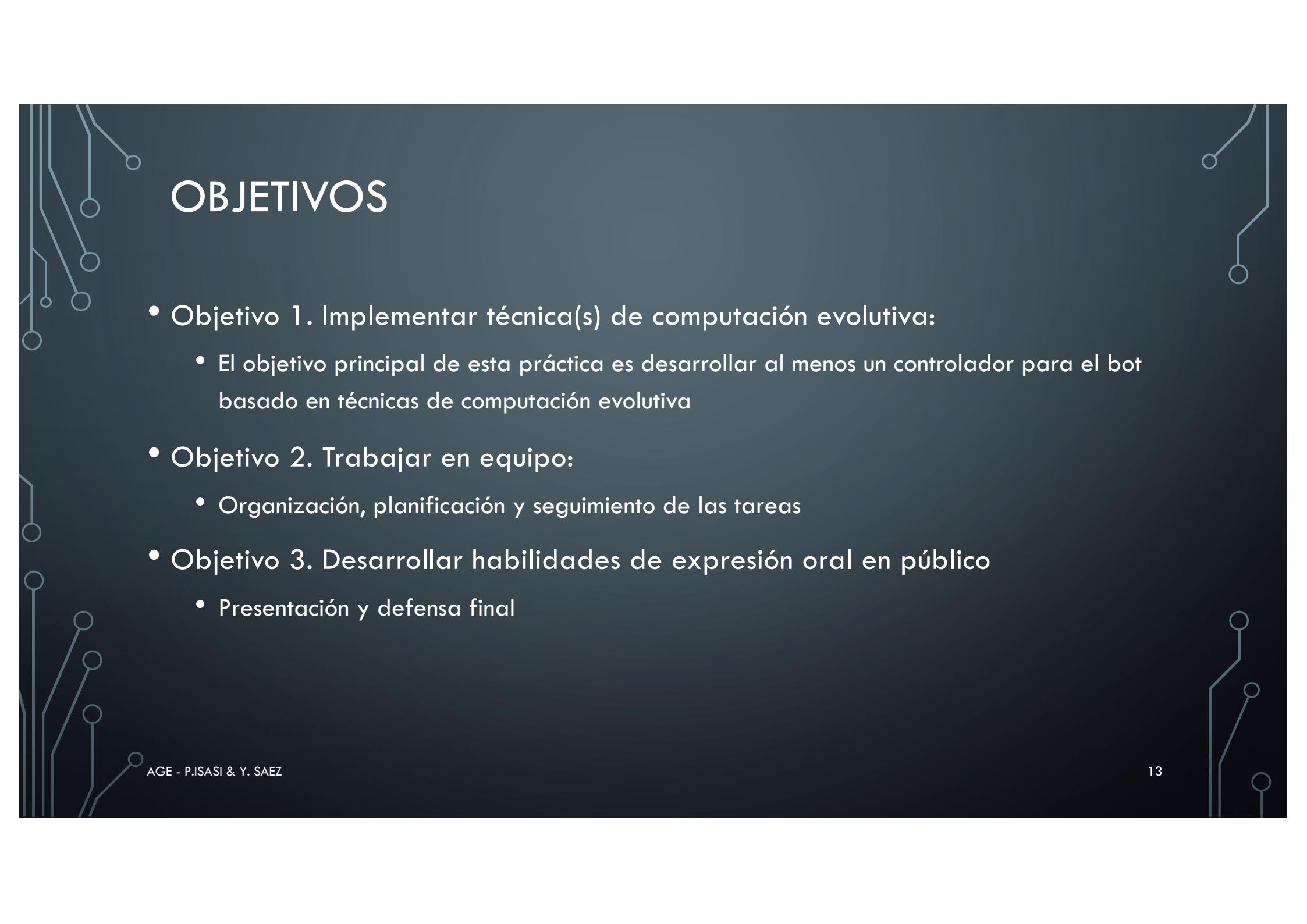
SIMULACIÓN



EJECUTAR MAIN.JAVA

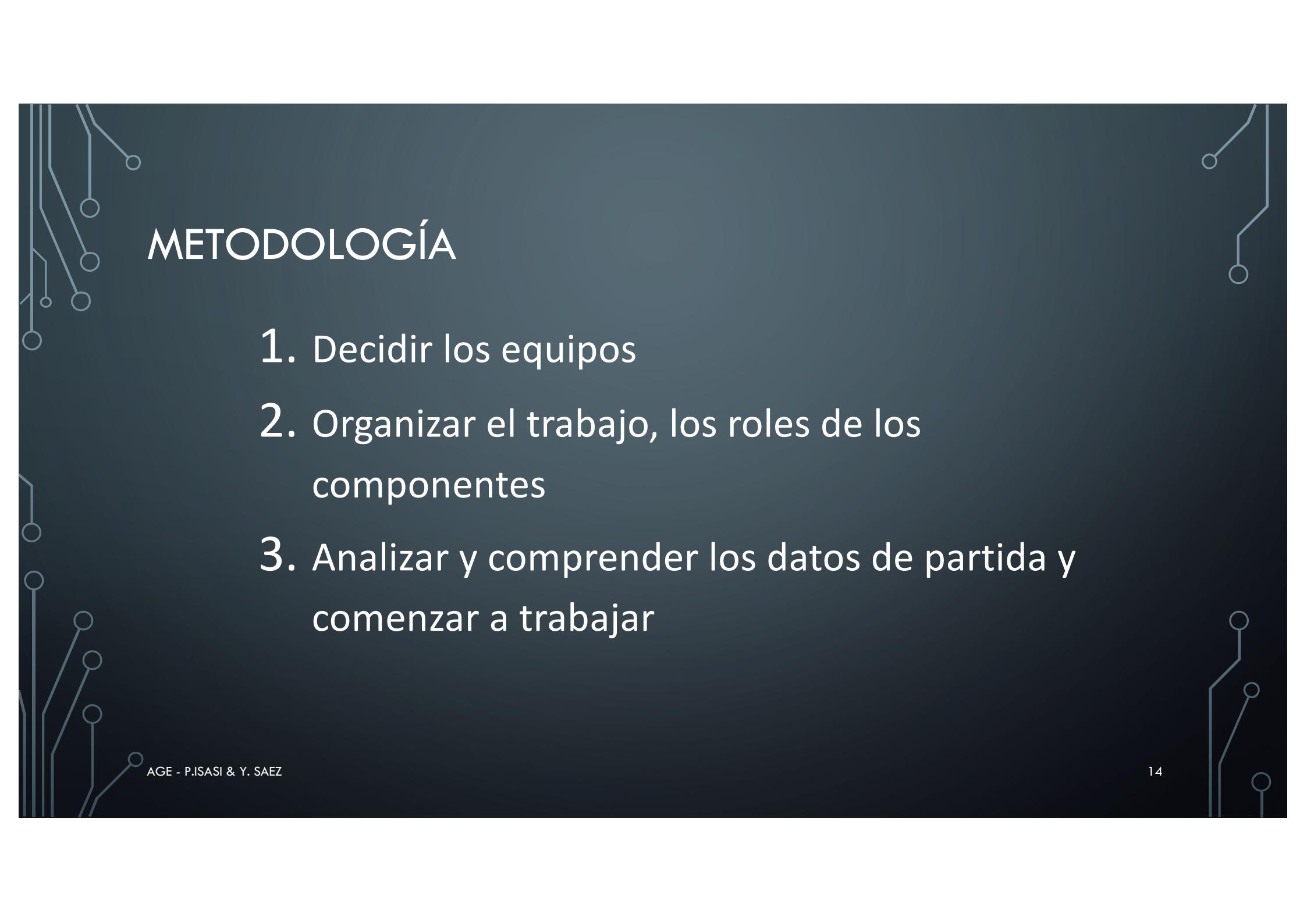


ACCEDER A
[HTTP://LOCALHOST:8888/TEST.HTML](http://localhost:8888/test.html)



OBJETIVOS

- **Objetivo 1. Implementar técnica(s) de computación evolutiva:**
 - El objetivo principal de esta práctica es desarrollar al menos un controlador para el bot basado en técnicas de computación evolutiva
- **Objetivo 2. Trabajar en equipo:**
 - Organización, planificación y seguimiento de las tareas
- **Objetivo 3. Desarrollar habilidades de expresión oral en público**
 - Presentación y defensa final

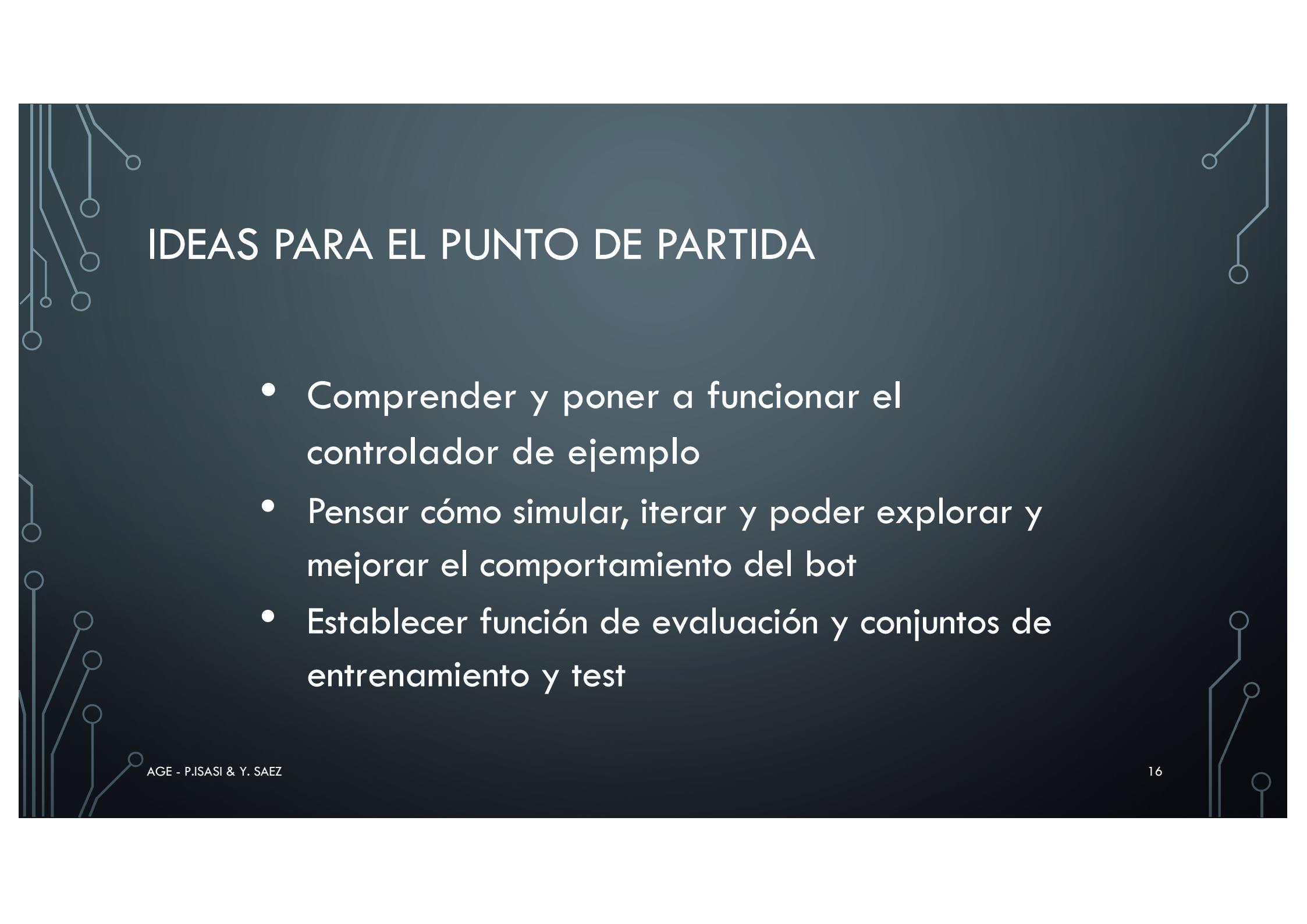


METODOLOGÍA

1. Decidir los equipos
2. Organizar el trabajo, los roles de los componentes
3. Analizar y comprender los datos de partida y comenzar a trabajar

PUNTO DE PARTIDA

1. Descargar el código del aula global
2. Instalar código en IDE preferido (java - maven)
3. Ejecutar main.java
4. Acceder a servidor virtual
5. Observar resultados, comprender comportamiento
AgentAGE1.java



IDEAS PARA EL PUNTO DE PARTIDA

- Comprender y poner a funcionar el controlador de ejemplo
- Pensar cómo simular, iterar y poder explorar y mejorar el comportamiento del bot
- Establecer función de evaluación y conjuntos de entrenamiento y test

EVALUACIÓN I

- El día 8 de diciembre se defenderá el desafío
- La defensa consistirá en una presentación oral de 15 minutos del trabajo realizado
- El mismo día de la presentación se hará entrega de la memoria y el código fuente (junto a las instrucciones para ejecutarlo)
- En la memoria* se debe incluir (al menos):
 - Planificación inicial y final
 - Roles designados a los componentes del equipo
 - Relación de tareas realizadas por cada miembro del equipo
 - Descripción de la técnica(s) implementadas
 - Justificación, ventajas e inconvenientes
 - Pruebas (gráficos, tablas, etc..)
 - Conclusiones
 - Anexo con comentarios acerca del trabajo en equipo

* No más de 20 hojas

EVALUACIÓN II

- La evaluación se realizará sobre (al menos) los siguientes aspectos:
 - Solución propuesta
 - Funcionamiento general del controlador y eficacia respecto a otras soluciones presentadas (ranking/turnos)
 - Presentación oral
 - Capacidad de organización del equipo
 - Competencias individuales adquiridas por cada miembro
 - Calidad de la memoria