**МИНОБРНАУКИ РОССИИ**

**Санкт-Петербургский государственный**

**электротехнический университет**

**«ЛЭТИ» им. В.И. Ульянова (Ленина)**

**Кафедра МОЭВМ**

отчет

**по практической работе №5**

**по дисциплине «Интеллектуальные системы»**

**Тема: «Решение целевой задачи в условиях противодействия»**

|  |  |  |
| --- | --- | --- |
| Студент гр. 8382 |  | Мирончик П.Д. |
| Преподаватель |  | Беляев С.А. |

Санкт-Петербург

2022

# Цель работы

Целью работы является решение задачи создания универсального механизма управления игроком в мультиагентной среде.

Для достижения поставленной цели необходимо решить следующие задачи:

- создать программу, обеспечивающую интерпретацию временного автомата;

- создать временные автоматы для игроков и вратаря;

- проверить забивание гола и защиту ворот от забивания гола (должны быть и успешные перехваты, и успешные голы).

# Ход работы

Для решения задачи было разработано дерево решений, аналогичное описанному в предыдущей работе. Распишем подробнее особенности реализации таймеров и связанных с ними переменных.

class Memory {

constructor() {

this.\_data = new Map()

}

increaseAge() {

for (let key of this.\_data.keys()) {

this.\_data.get(key).age++

}

}

updateValue(key, value) {

this.\_data.set(key, {

value: value,

age: 0

})

}

getValue(key) {

return this.\_data.get(key)?.value

}

getAge(key) {

return this.\_data.get(key)?.age ?? 10000

}

has(key) {

return this.\_data.has(key)

}

delete(key) {

this.\_data.delete(key)

}

}

Класс Memory предоставляет весь необходимый функционал для работы с временными переменными. Он позволяет создать новую переменную с нулевым таймером, увеличить значение таймера для всех переменных (что происходит каждый раз при смене такта), получить значение и «возраст» (значение таймера) переменной, удалить переменную или проверить ее наличие.

Использование такого таймера достаточно простое. Рассмотрим его на примере реализации алгоритма «запоминания» мяча.

class BallAnalyzer {

constructor(agent, mem) {

this.agent = agent

this.mem = mem

this.\_wasKicked = false

if (!this.\_hasBall() || this.\_getBallAge() > 0)

this.\_update()

}

isReady() {

return this.\_hasBall()

}

wasKicked() {

return this.\_wasKicked

}

estimateCoords(depth) {

return this.\_estimateCoords(depth)

}

estimateVelocity(depth) {

return this.\_estimateVelocity(depth)

}

getAge() {

return this.\_getBallAge()

}

canKick() {

let playerSize = this.agent.params.player\_size

let kickableMargin = this.agent.params.kickable\_margin

return Coords.distance(this.estimateCoords(0), this.agent.position.coords) < kickableMargin + playerSize / 2

}

\_update() {

if (!this.agent.hear.isPlayOn() && !this.agent.hear.isKickOffAlly()) this.\_clear()

this.\_wasKicked = false

let oldEstimate = this.isReady() ? this.estimateCoords(5) : null

let visibleBallInfo = null

for (let obj of this.agent.position.objects) {

if (obj.isBall) {

visibleBallInfo = obj

break

}

}

let position = Utils.calculateObjectPositioning(this.agent, visibleBallInfo)

if (!position) return

let info = {

coords: position.coords,

velocity: this.\_calculateVisibleVelocity(visibleBallInfo)

}

this.\_updateBall(info)

if (oldEstimate !== null) {

let newEstimate = this.estimateCoords(5)

this.\_wasKicked = Coords.distance(oldEstimate, newEstimate) > 4

}

}

\_calculateVisibleVelocity(visibleBallInfo) {…}

\_clear() {

this.mem.delete("BallAnalyzer.ball")

}

\_updateBall(ball) {

this.mem.updateValue("BallAnalyzer.ball", ball)

}

\_hasBall() {

return this.mem.has("BallAnalyzer.ball")

}

\_getBall() {

return this.mem.getValue("BallAnalyzer.ball")

}

\_getBallAge() {

return this.mem.getAge("BallAnalyzer.ball")

}

\_estimateVelocity(depth) {…}

\_estimateCoords(depth) {…}

}

Класс BallAnalyzer является вспомогательным контроллером, который принимает на вход экземпляр агента (для определения текущей позиции агента, получения списка видимых объектов и т.д.) и экземпляр «памяти» класса Memory.

При инициализации этот контроллер проверяет возраст сохраненного в память мяча. Если мяч был создан не на текущем такте, значит его нужно, по возможности, обносить. Для этого мяч ищется среди видимых объектов и, если обнаруживается, то записывается в память вместо старого. Это позволяет «запоминать» позицию мяча и использовать ее, даже если мяч в данный момент находится вне зоны видимости.

При этом «возраст» мяча также используется для предсказания позиции мяча через определенное количество тактов. Это полезно, если в последний раз была известна скорость движения мяча. Так, если мяч последний раз был виден пять тактов назад, и имел скорость в 2 метра в такт, то мы можем достаточно точно предсказать, где он находится сейчас и где будет находиться еще через несколько тактов; для этого нам понадобится параметр ball\_decay, что приходит в одном из сообщений сервера и вектор скорости мяча. Используя эти значения, можно с применением формулы суммы геометрической прогрессии найти за константное время позицию мяча в любой момент времени в будущем. Экспериментальные данные показывают, что позиция предсказывается достаточно точно в пределах полуметра при дальности перемещении мяча на 10 метров – это весьма неплохой результат, если учесть, что на него накладываются также погрешности в измерении направления взгляда, измерении скорости, вычисления позиции агента и прочее.

Примерно аналогичный способ запоминания данных можно применять и к игрокам на поле. Так, позиции союзных игроков запоминаются и используются в пределах пяти тактов с момента последнего визуального контакта с игроками. Это позволяет, например, отдавать достаточно точные пасы, даже не видя союзного игрока.

Для обеспечения лучшего качества перехвата было также реализовано запоминания последнего принятого решения по перехвату. Если игрок решил, что мяч следует перехватить в определенной точке, то, при продолжении перехвата, он будет двигаться к этой точке в течение следующих 5 тактов, не выполняя перерасчет оптимальной траектории перехвата на каждом из тактов. Несмотря на кажущуюся непрактичность данного действия (неужели не лучше было бы корректировать курс на каждом такте?), такой подход позволяет избавиться от замирания игрока на месте, когда агент стоит в одной точке и, после пересчета курса, выполняет поворот туловища на каждом такте. Особенно заметна такая проблема становится на близких расстояниях, когда изменение угла до мяча происходит слишком быстро.

# Выводы

В ходе выполнения лабораторной работы был реализован временной автомат, позволяющий выполнить поставленные задачи. Агент получил возможность определять с определенной погрешностью позицию мяча даже после потери того из поля зрения, запоминать позиции игроков на определенное время и использовать «память» для запоминания курса движения, что позволило сделать движение игрока более быстрым в условиях используемой системы симуляции.