Министерство образования Республики Беларусь

Учреждение образования

БелорусскиЙ государственный университет

информатики и радиоэлектроники

Факультет информационных технологий и управления

Кафедра вычислительных методов и программирования

ОТЧЕТ

по лабораторной работе

**«Смешивание моделей поведения»**

|  |  |  |
| --- | --- | --- |
| Студент гр. 924403 |  | К.Д. САНДАРОВИЧ |
| Проверил |  | С.О. ЛУКАШОВ |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |

Минск 2022

# Общая постановка задачи

## 1. Краткие теоретические сведения

Приемы смешивания позволяют объединять и смешивать модели поведения без создания новых сценариев, если необходимо придать агенту гибридное поведение.

2. Системные подходы к решению задачи

Рассмотрим несколько способов смешивания моделей поведения. Разберем отличия, плюсы и минусы каждого смешивания.

3. Разбор решения поставленной задачи

1. Смешивание моделей поведения с весовыми коэффициентами.

Смешивание моделей поведения с весовыми коэффициентами чаще других используется для смешивания моделей поведения благодаря его возможностям и простоте реализации.

Прежде всего необходимо добавить в класс AgentBehaviour свойство weight со значением весового коэффициента и, в данном случае, присвоить ему значение 1.0f. Кроме того, требуется изменить функцию

Update, чтобы она передавала свойство weight в функцию SetSteering класса Agent.

Весовые коэффициенты используются для усиления управляющего воздействия steering перед добавлением в основную структуру управления.

Сумма весовых коэффициентов не должна превышать 1.0f. Параметр weight является ссылкой и определяет значимость управления steering относительно прочих.

2. Смешивание моделей поведения по приоритету.

Иногда смешивание с учетом весовых коэффициентов не дает нужного результата, поскольку тяжеловесные модели поведения нивелируют влияние легковесных, в то время как эти модели поведения также должны вносить свой вклад в результат. В этом случае применяется смешивание по приоритетам, обеспечивающее каскадный эффект, от высоко- до низкоприоритетных моделей.

Этот подход очень похож на используемый в предыдущем рецепте. Нужно добавить новое свойство в класс AgentBehaviour и изменить функцию Update, организовав передачу свойства приоритета priority в функцию SetSteering класса Agent.

При создании групп по приоритетам смешиваются совместимые модели поведения, а затем выбирается первая группа, в которой величина управляющего воздействия превышает пороговое значение. В противном случае выбирается значение из группы с наименьшим приоритетом.

3. Комбинирование моделей поведения с применением конвейера управления.

Это еще один подход к смешиванию моделей поведения, основанный на целях. Он является золотой серединой между смешением перемещений и планированием без реализации последнего.

При использовании конвейера изменяется сам подход к управлению. Здесь в центре внимания находятся цели и ограничения. То есть вся тяжесть осуществления ложится на базовые и производные от них классы, определяющие модель поведения.

Класс SteeringPipeline использует реализованные ранее классы, обеспечивая управляющий компонентами конвейер.

Этот код конструирует составную цель из списка целей targeters, создает дочерние цели с помощью деструкторов из списка decomposers и перед включением их в конечную цель проверяет соответствие ограничениям из списка constraints. Если ни одна из целей не прошла проверку, используется модель поведения по умолчанию Wander.

# Листинг

public void SetSteering (Steering steering, float weight)

{

this . steering . linear += (weight \* steering . linear) ;

this . steering . angular += (weight \* steering . angular) ;  
}

private Steering GetPrioritySteering ( )

{

Steering steering = new Steering ( ) ;

float sqrThreshold = priorityThreshold \* priorityThreshold;

foreach (List<Steering> group in groups . Values)

{

steering = new Steering ( ) ;

foreach (Steering singleSteering in group)

{

steering . linear += singleSteering . linear;

steering . angular += singleSteering . angular;

}

if (steering . linear . sqrMagnitude > sqrThreshold ||

Mathf .AЬs (steering . angular) > priorityThreshold)

{

return steering;

}

}

public override Steering GetSteering ( )

{

Goal goal = new Goal ( ) ;

foreach (Targeter targeter in targeters )

goal . UpdateChannels (targeter . GetGoal ( ) ) ;

foreach (Decomposer decomposer in decomposers )

goal = decomposer . Decompose (goal ) ;

for ( int i = О; i < constraintSteps ; i++)

{

Path path = actuator . GetPath (goal) ;

foreach (Constraint constraint in constraints)

{

if (constraint . WillViolate (path) )

{

goal = constraint . Suggest (path) ;

break;

}

return actuator . GetOutput (path, goal ) ;

}

}

return base . GetSteering ( ) ;

}

# Вывод

В процессе выполнения лабораторной работы были реализованы системы смешивания моделей поведения.