河南科技学院

2022 届本科毕业论文（设计）

英文文献及翻译

Conceptualization and Implementation of a Reinforcement Learning Approach Using a Case-Based Reasoning Agent in a FPS Scenario

概念化和实施强化学习方法使用 FPS 场景中基于案例的推理代理

学生姓名：杨勐奇

所在院系：信息工程学院

所学专业：计算机科学与技术

评阅意见：通过 导师签名：黄雪

原文:

Conceptualization and Implementation of a Reinforcement Learning Approach Using a Case-Based Reasoning Agent in a FPS Scenario

Marcel Kolbe1 , Pascal Reuss1,2 , Jakob Michael Schoenborn1,2 , and Klaus-Dieter Althoff1,2

Abstract: This paper describes an approach that combines case-based reasoning (CBR) and reinforcement learning (RL) in the context of a first person shooter (FPS) game in the game mode deathmatch. Based on an engine written in C#, Unity and a simple rule-based agent, we propose a FPS agent who is using a combination of case-based reasoning and reinforcement learning to improve the overall performance. The reward function is based on learned sequences of performed small plans and considers the current win chance in a given situation. We describe the implementation of the reinforcement algorithm and the performed evaluation using different starting case bases.

Keywords: Case-Based Reasoning · Reinforcement Learning · First Person Shooter · Multi-Agent System · Planning

1. Introduction and motivation

To prove the functionality of an artificial intelligence, multi-agent systems became a popular application area. We take a look at the first person shooter (FPS) domain, a sub-genre of action video games. In a typical FPS game, there are two teams of typically human players trying to overcome the opposing team by either eliminating each member of the opposing team or by successfully complete another objective, such as planting a bomb at a certain place or by preventing the opposing team to do so. While both teams are actively playing at the same time, most FPS are limited by a round-time of approximately five minutes and a limited map size. Each individual FPS game differentiates itself from other games by having certain unique characteristics, for example, the extent of reality. However, a FPS game can be generalized to the following: The human player takes over the sight of an agent, as if he or she would be placed inside of the game. To emphasize this, environmental sounds, for instance, the sound of a step when moving through the terrain, are implemented to increase the immersion into the game. An agent starts with a certain level of health points (HP, usually 100) and with a basic pistol weapon equipped, having a limited amount of ammunition. The goal is to find and eliminate the opposing agent. Each round, credits are earned based on the outcome of the last round to buy better weapons or supply like armor and health packs to increase the current health points. This domain is not limited to one versus one fights but rather is usually applied to five versus five combat. This increases the complexity a lot since factors like communication and planning between the agents have to be considered: while the game is running, the current state changes each millisecond by usually not knowing the enemies position and by creating predictions of the enemies next possible steps. Since it is a very advantageous situation to see the enemy before the enemy sees you, one of the most common tactics to gain this advantage is to “camp”, that is, waiting and hiding for an indefinite amount of time3 at a point of interest until the enemy passes along. However, especially in the currently most played FPS games like Fortnite or Players Unknown Battleground, the so-called mode “Battle Royale” mode gained an increasing popularity. This mode is usually a free-for-all mode, meaning everyone fights for himself against any other agent4 . Using this mode, after a certain amount of time, the size of the level shrinks until no place is left to hide and the last two surviving agents will have to engage each other. The longer one can avoid fights and look for equipment, the better. Thus, an AI agent would need to plan accordingly when a rather defensive behavior has to be switched to a comparably aggressive behavior. Since the spawn locations and spawn timings of these equipment differ each round, each game the player is able to experience a new situation. This is where our research adds on.

While CBR and RL were and are used in different game domains (see section 2), the use of CBR and RL in a FPS game in the game mode of deathmatch, either one-on-one or team-based is new and has different strategies, tactics and challenges than game modes like conquest or capture the flag. Another goal of or research is to

apply the CBR and RL approach to FPS in the game mode deathmatch and evaluate the progress of our agents during the games. As we will later see more in detail in section 3.1, one might think of one round of a FPS game as one case which can be stored and retrieved. We went on a more detailed level and used the current perception of the agent as situation and mapped a corresponding action as a solution to the current situation. In our first version, we used 17 attributes, such as currentAmmunition, distanceToEnemy, among many others and 15 initial cases on a FPS game designed in Unity [9]. However, the initial problem space was not large enough, leading to a unique overall dominant tactic by always looking to pick up the weapon upgrade and therefore nullifying the benefits of CBR. While we increased the complexity of the level itself, we ask the research question whether we can improve the retrieval of the best cases by using reinforcement learning to prevent the agent from being stuck in a corner, not changing his situation and thus not retrieving another case to get himself out of the corner.

For our own testing purposes, we used a FPS game developed by Jannis Hillmann as part of his master's thesis using a combination of Unity, C# and JAVA [5]. Fig. 1 shows an example of a CBR agent (purple) fighting a rule-based AI (orange). This platform in general is intended to support other domains, for example, real-time strategy games and economical simulations to name a few.

To enable another learning component besides case-based reasoning for our agent, we propose to use reinforcement learning (RL). Using the definition from Sutton, RL enables to derive actions from situations. Situations are mapped to an aim such that actions are connected with certain rewards or punishments (negative rewards) [10]. During each situation, the agent has to perceive the current state of the environment he is situated in. One of the advantages of reinforcement learning is that not only the current situation but also near-future situations can be considered. Combining the lessons learned from failed experiences and the positive experiences from properly rewarded situations leads to the most important aspect of RL and consequently to the creation of an intelligent plan during a reasonable time frame.

2. Related Work

RL has been used in gaming AI before mainly in the context of real time strategy (RTS) games. One approach was developed by Wender and Watson in StarCraft II [14]. StarCraft II as a RTS game has almost the same requirements and conditions, e. g., real-time decision making with incomplete information, as a FPS game but with more focus on macro- and micromanagement. Thus, the domains are to some extent interchangeable and approaches can be transferred by applying the necessary adjustments since the focus of the two genres are different. Using reinforcement learning in combination of case-based reasoning, we want to prove that a learning agent can always defeat a rather static rulebased agent after enough experiences have been collected and thus lead to a beneficial approach. According to Wender and Watson, the use of different forms of CBR and RL in the context of AI research is one of the most common ways of working, regardless of whether the methods work together or act separately. The two approaches support each other's respective problem areas. [14]

There are many other approaches that combine CBR and RL in RTS games. The approach of [13] combines goal-driven autonomy with RL to coordinate the learning efforts of multiple agents in a RTS games by using a shared reward function. In [11] a generalized reward function is used to apply several RL algorithms to a RTS game, while [12] propose two RL algorithms bases on neural networks to optimize the build order in StarCraft II. In the last years a new approach called Deep RL was developed and used in several research work like [1], [3], [4] and [7]. All these approaches uses CBR and RL to improve the micro and/or macro management in RTS games, but RL can also be used in turn-based games [15] and in FPS games. The work of Auslander and his colleagues [2] deals with the use of a RL logic in the context of a first person shooter Unreal Tournament, developed with the UNREAL Engine. Their work is based on the previous successes of various knowledge-based learning approaches that have relied on the mechanics and dynamics of games, e. g. Star Craft or Backgammon. Instead of learning "one-on-one" confrontation, the work describes group-oriented behaviors and strategies based on the combination of RL and the CBR approach for the game Unreal Tournament. This is a challenge for Unreal Tournament as different game modes with different goals are supported. On the one hand it can be a team-based deathmatch and on the other hand it can be a game type called conquest, in which the killing of opponents is secondary. The ultimate goal is to defend a specific point on the map. According to [2], the game mode conquest is particularly interesting for the development of cooperative strategies in the context of a team-oriented approach. Furthermore, it offers a good opportunity to observe the learning behaviour of the team.

Another work in the context of a FPS game is presented by [8] for the MineCraft game. The goal of this game is not to fight against each other like in Unreal Tournament or our approach, but to harvest resources and build structures. The approach combines hierarchical task networks and RL algorithms to adapt the plans for agents acting in the dynamic world. A deep RL approach for the FPS game Doom was developed by [6].

3. Architecture and structural dependencies

Every agent and the environment is built in Unity 3D and thus implemented as a C# project. We use three different agents: Player Agent, Planning Agent, Communication Agent. The Player Agent uses an inherited method called update() which updates the agent perception, i.e., the agent received information throughout its sensors or proposed plans of the planning agent. As the update()-method triggers multiple times per second, this value has to be evaluated considering the fairness towards a human enemy who can only evaluate a limited amount of perceptions. With each update()-cycle, the agents sends a request to the Communication Agent. This agent is connected with the myCBR component which uses JAVA as a programming language. Thus, a corresponding interface using the communication via TCP/IP has been established. Each time the Communication Agent receives a request, the agent formulates a request to the CBR system to retrieve the most similar case to the current situation. Once the most similar case has been retrieved, the proposed solution (which usually results in a proposed action) will be sent to the Planning Agent. This agent evaluates the proposed action from the Communication Agent and forms a plan,

which will be sent to the Player Agent and will be followed until a new plan will be proposed. As a simple example, the Player Agent perceives that he is low on health (e.g., < 20 % hit points (HP)), so that perception will be transferred to the Communication Agent. This agent retrieves as the most similar case, that picking up a piece of pizza leads to gain back most of the lost hit points. Thus, the planning agent formulates a plan to find and to pick up a piece of pizza. This plan will be followed, until it has been picked up, until the agent dies, or until another plan gains a higher priority (e.g., self-defence or perceiving another, better, healing item). The structure builds up as described can be seen in Fig.

译文：

概念化和实施强化学习方法使用 FPS 场景中基于案例的推理代理

马塞尔·科尔贝, 帕斯卡·罗伊斯, 雅各布·迈克尔·舍恩伯恩和
克劳斯·迪特阿尔托夫

摘要：本文描述了一种游戏模式为死亡竞赛的第一人称射击 (FPS) 游戏机制, 上下文中结合基于案例推理 (CBR) 和强化学习 (RL) 的方法。基于用 C#、Unity 编写的引擎和一个简单的基于规则的代理，我们提出了一种 FPS 代理，它结合使用基于案例的推理和强化学习来提高整体性能。奖励函数基于已执行计划的学习序列，并考虑给定情况下的获胜机会。我们描述了强化算法的实现以及使用不同的起始案例库进行评估。

关键词：案例推理, 强化学习, 第一人称射击, 多智能体系统, 规划设计

1. 介绍

为了证明人工智能的能力，多智能体系统成为一个流行的应用领域。我们来看看第一人称射击游戏（FPS）领域，动作视频游戏的子类型。在典型的 FPS 游戏中，有两个典型的人类玩家组成的团队试图通过以下方式战胜对方玩家消灭对方队伍的每个成员或通过成功完成另一个目标，例如在某个地方放置炸弹或阻止对方玩家这样做。在两支队伍同时比赛的同时，大多数 FPS 受限于大约五分钟的时间和有限的地图大小。每个单独的 FPS 游戏都通过具有某些独特的特征（例如，现实的程度）与其他游戏区分开来。然而，FPS 游戏可以概括为：在玩家的视线范围内，就好像他或她将被置于游戏中一样。为了强调这一点，环境声音，例如，当脚步声在地形中移动，以增加对地形的沉浸感游戏。一个特工开始时有一定的生命值（HP，通常是 100）和配备基本手枪武器，弹药数量有限。目标是找到并消灭对手。每一轮，学分是根据上一轮的结果获得购买更好的武器或补给像盔甲和健康包来增加当前的健康点。这个领域不限于一对一的战斗，而是通常适用于五对五的战斗。这大大增加了复杂性，因为必须考虑玩家代理之间的沟通和计划等因素：虽然游戏是运行的时候，状态每毫秒更改一次，且通常不知道敌人的位置，并通过创建敌人的下一个可能步骤的预测。因为先见敌之后是非常有利的局面，获得这种优势的最常见策略之一是在固定点等待直到敌人过去。但是，尤其是在目前玩得最多的 Fortnite 或 Players Unknown Battleground 等 FPS 游戏，即所谓的“大逃杀”模式越来越受欢迎。

这种模式通常是免费模式，这意味着每个人都为自己而战，对抗任何其他特工。使用此模式，经过一定时间后，关卡的大小会缩小，直到无处可藏，最后两名幸存的特工将不得不各自交战其他。避免打架和寻找装备的时间越长越好。因此，当一个相当防御性的行为发生时，人工智能代理需要做出相应的计划被切换到一个比较激进的行为。由于复活地点并且这些设备的生成时间每轮都不同，每场比赛的玩家能够体验新的情况。这是我们的研究的地方。

虽然 CBR 和 RL 曾经并且被用于不同的游戏领域（参见章节 2）、死亡竞赛游戏模式下 FPS 游戏中 CBR 和 RL 的使用，一对一或团队合作都是新事物，具有不同的战略、战术和挑战比征服或夺旗等游戏模式。的另一个目标或者研究是在游戏模式下将 CBR 和 RL 方法应用于 FPS 死亡竞赛并评估我们的特工在比赛期间的进展。

正如我们稍后将在 3.1 节中更详细看到的那样，人们可能会想到一轮 FPS 游戏的一个案例，可以存储和检索。我们进行了一次更详细的分级，并使用代理

当前感知作为情境和相应的行动映射为当前情况的解决方案。在我们的第一个版本中，我们使用了 17 个属性，例如 currentAmmunition、distanceToEnemy、在 Unity 中设计的 FPS 游戏中的许多其他案例和 15 个初始案例。然而，最初的问题空间不够大，导致了一种独特的整体优势策略，总是寻求拿起武器升级，因此抵消了恢复血量的好处。虽然我们增加了复杂性水平本身，我们研究问题是否可以提高检索通过使用强化学习来防止代理被困在角落里，没有改变他的处境，因此没有检索另一个案件让自己走出角落。

为了我们自己的测试目的,我们使用了 Jannis 开发的 FPS 游戏 Hillmann 作为他硕士论文的一部分，使用 Unity、C# 和 javascript。图 1 显示了一个 CBR 代理（紫色）对抗基于规则的示例人工智能（橙色）。该平台通常旨在支持其他领域，例如，实时战略游戏和经济模拟等等。

为我们的基于案例的推理启用另一个学习组件代理，我们建议使用强化学习（RL）。使用来自的定义 Sutton, RL 能够从情境中得出行动。情况映射到行为与某些奖励或惩罚相关联的目标（负奖励）。在每种情况下，代理都必须感知他所处环境的当前状态。强化学习的优点之一是，不仅是当前的情况，而且是近期的情况可以考虑。结合从失败经验中吸取的教训和从适当奖励情况中获得的积极经验，可以得出 RL 最重要的方面，因此对创建智能在合理的时间范围内制定计划。

2. 相关工作

RL 之前主要用于实时策略的游戏 AI 中(RTS) 游戏。星际争霸 II 作为 RTS 游戏具有几乎相同的要求和条件，例如信息不完整的实时决策，作为 FPS 游戏，但更注重宏观和微观管理。就这样领域在一定程度上可以互换，方法可以转移通过应用必要的调整，因为这两种类型的重点是不同的。结合基于案例的推理使用强化学习，我们想证明在收集到足够的经验后，学习代理总是可以击败一个相当静态的基于规则的代理,从而导致有益的做法。根据 Wender 和 Watson 的说法，不同形式的使用人工智能研究背景下的 CBR 和 RL 是最常见的方法之一的工作，无论这些方法是一起工作还是单独行动。这两种方法相互支持各自的问题领域。

在 RTS 游戏中还有许多其他方法可以结合 CBR 和 RL。[13] 的方法将目标驱动的自主性与 RL 相结合以协调通过使用共享奖励函数来学习 RTS 游戏中多个代理的努力。在 [11] 中，一个广义的奖励函数用于应用几种 RL 算法到 RTS 游戏，而 [12] 提出了两种基于神经网络的 RL 算法优化星际争霸 II 中的构建

顺序。在过去的几年里，一种新的方法称为 Deep RL 被开发并用于多个研究工作，如 [1]、[3]、[4]和 [7]。所有这些方法都使用 CBR 和 RL 来改进微观或 RTS 游戏中的宏观管理，但 RL 也可用于回合制游戏[15] 和 FPS 游戏。 Auslander 和他的同事 [2] 的工作涉及在第一人称射击游戏 Unreal Tournament 的上下文中使用 RL 逻辑，使用虚幻引擎开发。他们的工作是基于以前各种基于知识的学习方法的成功，这些方法依赖于游戏的机制和动力学，例如星际争霸或双陆棋。这对 Unreal Tournament 来说是一个挑战，因为它支持具有不同目标的不同游戏模式。一方面它可以是基于团队的死亡竞赛，另一方面，它可以是一种游戏类型，称为征服，其中杀死对手是次要的。最终目标是保卫地图上的特定点。根据 [2]，游戏模式征服对发展合作战略特别感兴趣以团队为导向的方法。此外，它提供了一个很好的机会观察团队的学习行为。

[8] 介绍了 FPS 游戏背景下的另一项工作我的世界游戏。这场比赛的目标不是像在 Unreal Tournament 还是我们的方法，但是为了收获资源和建造结构。该方法结合了分层任务网络和 RL 算法调整在动态世界中行动的代理的计划。一种深度强化学习方法 FPS 游戏 Doom 由 [6] 开发。

3. 架构和结构依赖

每个代理和环境都是在 Unity 3D 中构建的，因此实现为 C# 项目。我们使用三种不同的代理：玩家代理、计划代理、通信代理。 Player Agent 使用一个名为 update() 的继承方法它更新代理感知，即代理通过其传感器或计划代理的建议计划接收信息。作为 update() 方法每秒触发多次，必须考虑到评估此值对只能评估有限数量的人类敌人的公平性。在每个 update() 循环中，代理向通信代理发送一个请求。此代理与 myCBR 组件连接，该组件使用 JAVA 作为编程语言。因此，相应的接口使用通过 TCP/IP 的通信已建立。每次通信代理收到请求时，代理都会向 CBR 提出请求系统检索与当前情况最相似的案例。曾经已检索到类似案例，建议的解决方案（通常会导致建议的行动）将被发送给规划代理。该代理评估通信代理提出的行动并形成一个计划，该计划将发送给玩家代理，并会一直跟进，直到提出新的计划。举个简单的例子，玩家代理感知到他的健康状况不佳（例如，< 20 % 生命值 (HP)），因此感知将转移到通信代理。该代理检索为最相似的情况，即拿起食用一块比萨饼可以恢复大部分失去的生命值。因此，规划代理人制定了一个寻找并拿起食用一块比萨饼的计划。这个计划将紧随其后，直到它被捡起食用，直到代理死亡，或直到另一个计划获得更高的优先级（例如，自卫或感知另一个，更好的治愈物品）。