10th International Scientific Conference
Technics, Informatics, and Education – TIE 2024
20-22 September 2024

Session: *Educational Technology*
Professional paper
DOI: 10.46793/TIE24.239A

Using Artificial Intelligence Concepts to Design Non-Playable Characters in Road Traffic Safety Games

Veljko Aleksić[1*] [0000-0003-2337-1288]
[1] University of Kragujevac, Faculty of Technical Sciences Čačak, Serbia
[*] veljko.aleksic@ftn.kg.ac.rs

Abstract: *The integration of artificial intelligence concepts into digital games design has revolutionized the gaming industry. Among other elements, artificial intelligence significantly influenced modern gameplay mechanics, elevated player experiences, and streamlined game development processes. Road traffic safety driving simulation games are an emerging educational tool aimed at improving road safety awareness and skills among drivers. A critical component of these games is the AI-driven Non-Playable Characters (e.g., NPCs) that expand dynamic and immersive gameplay experience by exhibiting various realistic road users' behavior patterns, traffic conditions and player actions adaptation. The adaptive AI algorithms ensure balanced difficulty, catering to gamers' diverse driving skill levels, while procedural content generation opened endless possibilities in designing game levels, environments, and tasks, enhancing game replayability and longevity. AI-powered virtual assistants can provide players with seamless in-game guidance, enhancing their engagement without disrupting the gameplay flow. Additionally, adaptable intelligent road traffic conditions can challenge players to strategize and adapt, contributing to more compelling and immersive gaming experiences. Contemporary software tools and engines streamlined game development processes and accelerated asset creation, bug detection, and playtesting. Automated game design processes, such as AI-driven level and procedural generation, expedited prototyping and iteration phases, while AI-driven analytics tools offered valuable insights into player behavior and preferences, enabling developers to optimize game mechanics and its content for maximum impact. The impact of artificial influence concepts on digital game design is poised to grow even further, promising exciting innovations and possibilities for future game designers and enthusiasts alike.*

Keywords: *digital games; traffic safety; artificial intelligence; NPCs*

1. INTRODUCTION

The digital gaming industry has experienced exponential growth and evolution, driven by technological advancements and creative innovations. Among the pivotal technologies influencing this sector, artificial intelligence (AI) currently stands out as a transformative force [1]. AI has not only revolutionized the way digital games are designed and developed but also how they are experienced by players. The global increase in road traffic accidents has underscored the need for effective formal, informal and non-formal educational tools to promote road safety as traditional methods of driver education often lack the practical and engaging elements necessary to prepare drivers for real-world scenarios. Road traffic safety games offer a dynamic and interactive platform for learning safe driving practices and understanding traffic regulations. This paper explores the impact of AI concepts on designing NPCs in road traffic safety games, as they play a pivotal role in creating realistic traffic environments that challenge players to apply their knowledge and skills. The behavior of NPCs must mimic real-world entities such as other drivers, pedestrians, and cyclists to provide authentic experiences. The integration of AI in the NPCs design enhances their realism by enabling the exhibition of complex behaviors and dynamical interaction with the player and the game environment. Historically, digital games have progressed from simple, rule-based systems to complex, interactive environments [2]. The early days of game design were marked by limited computational power and rudimentary AI, which restricted the behavior of NPCs and the dynamism of game worlds. However, the integration of sophisticated AI algorithms and machine learning techniques has enabled the creation of more intelligent, adaptive, and realistic game objects and elements. These advancements have led to significant improvements in NPCs behavior, procedural content generation, adaptive gameplay, and immersive game environments [3].

One of the most noticeable impacts of AI in game design is the evolution of NPCs. Modern AI-driven NPCs exhibit lifelike behaviors, making them more

than just scripted entities. They can learn from player actions, adapt their strategies, and interact with the game environment in a believable manner. This level of sophistication contributes to a more immersive and engaging gaming experience, as players can interact with NPCs in ways that mirror real-world interactions [4]. Moreover, AI has enabled the development of adaptive gameplay, where the game adjusts its difficulty, paths and challenges based on the player's level of skills and preferences [5]. This personalization ensures that games remain challenging and enjoyable for a wide range of gamers, from novices to experts. AI-driven adaptive systems analyze player behavior in real time, providing a tailored gaming experience that maintains a balance between challenge and reward. Procedural content generation (e.g., PCG) is another area where AI has made a substantial impact. By leveraging algorithms and machine learning, developers can create vast and varied game worlds without manually designing each element. PCG not only saves development time but also enhances replayability, as players can explore new content in each playthrough [6]. Games like *Forza Horizon*® exemplify the potential of PCG in creating expansive, dynamic, photo-realistic environments, as presented in Fig. 1.

Figure 1. *Forza Horizon 5 AI PCG environment*

AI has also contributed to the creation of intelligent game environments that respond to player actions and decisions [7]. These environments use AI algorithms to simulate realistic physics, dynamic weather conditions, and interactive elements, providing a more immersive and engaging experience [8]. As a result, players can experience a game world that feels alive and responsive, enhancing the overall sense of immersion.

2. AI MODELS IN DESIGNING NPCs

NPCs are fundamental components of digital games, contributing significantly to the game's narrative, environment, and player experience [9]. Historically, NPCs were driven by a simple rule-based system that limited their behavior and interaction capabilities. However, AI integration has revolutionized NPC control, enabling more dynamic, intelligent, and responsive characters. Road traffic safety games aim to simulate real-world driving conditions and traffic scenarios to educate players about safe driving practices and traffic regulations. NPCs in these games represent various road users and entities, creating a realistic traffic environment that challenges players to navigate safely and make informed decisions. In the early stages of digital game development, NPC behavior was largely predefined and scripted. Simple finite state machines (FSMs) were commonly used, which allowed NPCs to transition between a limited number of states based on specific conditions [10]. Implementing an FSMs involves defining the states, transitions, and actions in a way that can be processed by the game's AI system. For example, an NPC driver might have states for "driving," "stopping at a red light," and "yielding to pedestrians". FSMs are straightforward to implement and can effectively simulate simple traffic behaviors. This typically requires coding the FSM logic into the game engine using scripting languages or integrated development environments (IDEs) [11]. While these systems were sufficient for basic interactions, they often resulted in predictable and repetitive behaviors, reducing the overall immersion and realism [12]. The introduction of AI into NPC control marked a significant shift from these rudimentary systems. AI techniques such as pathfinding algorithms, decision trees, and machine learning models provided NPCs with the ability to make more complex decisions, adapt to player actions, and exhibit lifelike behaviors [13]. This evolution has been instrumental in creating more engaging and immersive gaming experiences.

2.1. Pathfinding Algorithms

Pathfinding algorithms are a crucial component in the design of NPCs in digital games, enabling characters to navigate complex environments efficiently and realistically [14]. Effective pathfinding algorithms enable NPCs to navigate complex traffic scenarios, avoid collisions, and adhere to traffic rules. Key metrics in pathfinding algorithms are *optimality* (finding the shortest or most efficient path), completeness (ensuring a path is found if one exists), and complexity (computational resources required to find the path). NPC control can be operationalized by several pathfinding algorithms:

- *Dijkstra's algorithm* computes the shortest paths from a starting node to all other nodes in a graph with non-negative weights [15]. While it guarantees finding the shortest path, it can be computationally intensive for large game environments. Dijkstra's algorithm is particularly useful in scenarios where NPCs need to follow specific routes or navigate through dense traffic networks, ensuring they take the most efficient routes to their destinations [16].
- *A* algorithm* combines the benefits of Dijkstra's algorithm and a heuristic approach. It is a widely

used technique, enabling NPCs to find the shortest path between points while avoiding obstacles [17]. The heuristic component enables efficient and realistic pathfinding, allowing units to move dynamically and strategically, particularly in complex and dynamic environments. NPCs can maneuver around obstacles and other units, reaching their destinations without unnecessary delays or collisions, thereby enhancing the strategic depth of the game. In road traffic games, the A* algorithm helps NPCs navigate the game environment by considering factors such as road layout, traffic signals, and the positions of other vehicles. The use-case scenario of hybrid A* algorithm implementation for self-driving NPC pathfinding in Unity environment is presented in Fig. 2.

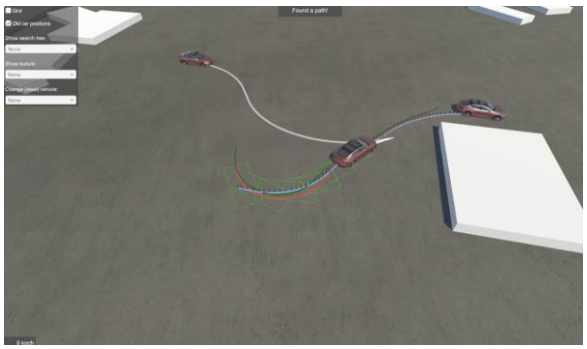

Figure 2. *Self-driving vehicle simulation using A* algorithm in Unity environment*

- *D* algorithm* is an extension of the A* algorithm designed for environments that change over time. In games with dynamic environments where obstacles can appear or disappear, D* ensures that NPCs can recalculate and adapt their paths, maintaining efficient navigation suitable for real-time applications [18].
- *Jump Point Search* (JPS) is an optimization of the A* algorithm for uniform-cost grids. It reduces the number of nodes evaluated by "jumping" over nodes that do not affect the final path, significantly improving efficiency [19]. Like A*, JPS uses a heuristic function to estimate the cost of reaching the goal from a given node. The heuristic function typically used in JPS is the Manhattan distance for grid-based environments, which calculates the distance between two points as the sum of the absolute differences in their x and y coordinates. JPS reduces the computational load of pathfinding by minimizing the number of nodes that need to be evaluated. This allows the game to support larger and more complex traffic environments without compromising performance. The reduced computational load also enables the game to handle more NPCs simultaneously, creating richer and more diverse traffic scenarios.
- *Navigation meshes* are a more advanced pathfinding technique where the areas of the environment are represented as interconnected polygons. Waypoints are predefined points in the game environment that guide NPC movement along specific paths. They are commonly used in games to define patrol routes or navigation paths for NPCs. Traffic flow models, such as the Cellular Automata Model, simulate the movement of vehicles and pedestrians by dividing the environment into a grid of cells, each representing a portion of the road [20]. NPCs move from one cell to another based on traffic rules and interactions with other NPCs, creating a realistic simulation of traffic flow. Navigation meshes can be easily scaled to handle large and complex environments, making them suitable for open-world games and large-scale simulations. For instance, the Unreal Engine development environment has a built-in navigation mesh system that provides tools for automatic mesh generation and real-time updates, simplifying the implementation of advanced navigation in games [21]. A traffic training simulator developed using Unreal Engine and NavMesh provides a realistic and interactive environment for driver training (Fig. 3). The simulation includes a complex road network with various traffic scenarios, such as congestion, road closures, and accidents. NavMesh enables the vehicles to navigate the environment smoothly and realistically, providing drivers with valuable training on safe driving practices.

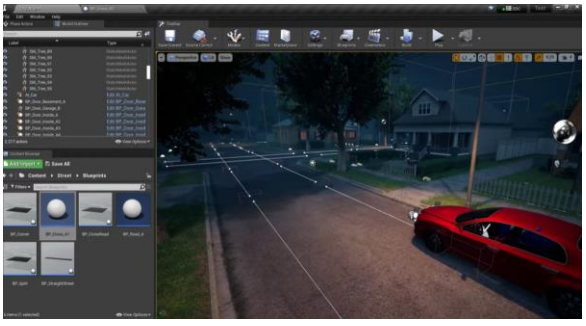

Figure 3. *Unreal Engine NavMesh NPC vehicle navigation*

A hierarchical pathfinding model is used for breaking down the environment into multiple levels of abstraction where the high-level paths are calculated first, followed by detailed paths at lower levels. Large-scale massively multiplayer online (MMO) games use hierarchical pathfinding for NPC optimization by simplifying complex environments into manageable sections, ensuring efficient and scalable pathfinding [22]. The flow fields technique provides a vector field across the environment that guides NPCs toward their goals. In cases where NPCs operate in close proximity to each other, a cooperative pathfinding technique is used to avoid collisions and optimize group movement.

2.2. Decision Trees and Behavior Trees

Decision tree algorithms are a versatile and powerful tool in the NPCs design. A decision tree is a flowchart-like structure where each internal node represents a decision point based on certain conditions, each branch represents the outcome of a decision, and each leaf node represents an action or end state [23]. This hierarchical structure allows for clear and logical decision-making processes. Implementing a decision tree involves defining the conditions and actions at each node and creating the logic to traverse the tree based on the current game state. This transparency allows developers to fine-tune AI behavior for a more challenging and realistic experience. Decision trees can be easily expanded by adding more nodes and branches, making them flexible to accommodate complex behaviors and scalable to handle a wide range of scenarios. This model also allows modular design, where individual branches or sub-trees can be developed and tested independently. The modularity simplifies development and facilitates the reuse of decision logic across different NPCs.

Behavior trees have become a popular framework for controlling NPCs due to their flexibility, modularity, and ease of understanding. Originating from robotics and AI research, behavior trees provide a structured way to model complex decision-making processes, enabling NPCs to exhibit realistic and adaptive behaviors. A behavior tree is a hierarchical model that represents the execution flow of an NPC's behavior. It consists of nodes, which can be of different types: control nodes (such as sequences and selectors) and execution nodes (tasks or actions) [24]. Control nodes manage the flow of execution. Sequence nodes execute the child nodes in order until one fails, while selector nodes execute the child nodes in order until one succeeds. Execution nodes perform specific actions or checks. They return to success, failure, or running states. For instance, the casual traffic sequence in an NPC car can be reused across different NPCs, each with its specific waypoints, by simply plugging in different action nodes. Rule-based systems use predefined rules to guide NPC behavior. These systems are particularly effective for simulating complex traffic interactions and ensuring that NPCs adhere to traffic laws. For example, rules can be defined for stopping at red lights, yielding to pedestrians, and maintaining safe following distances. Rule-based systems provide a straightforward way to implement traffic regulations in the game. Behavior trees are highly flexible and can be expanded or modified with minimal impact on the overall structure. They scale well with increasing complexity, making them suitable for games with complex NPC behaviors. An NPC's behavior tree can be expanded to include additional sequences or selectors for new actions. Tools like Unreal Engine's Behavior Tree editor provide a visual representation of the tree, allowing designers to intuitively adjust the flow of behaviors without deep programming knowledge (Fig. 4).

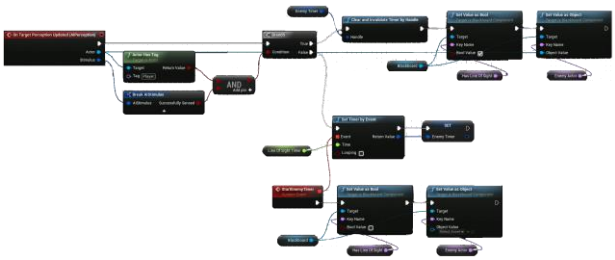

Figure 4. *The example of Unreal 4 behavior tree NPC control*

For instance, the Far Cry game series relies heavily on behavior trees to control both enemy and wildlife NPCs. These trees enable NPCs to exhibit a wide range of behaviors, from patrolling and engaging in combat to hunting and fleeing, contributing to the series' immersive open-world gameplay.

2.3. Fuzzy Logic

Traditional approaches often struggle with the inherent uncertainties and variabilities in driving environments and behaviors. Fuzzy logic, with its ability to handle imprecise information and model human-like reasoning, offers a robust solution for enhancing driving simulations. Fuzzy logic is an approach to computing that handles the concept of partial truth, with truth values ranging between completely true and completely false. In the NPC context, fuzzy logic algorithms allow for more nuanced decision-making processes, enabling NPCs to exhibit realistic and adaptive behaviors. Fuzzy logic extends classical binary logic to handle the concept of partial truth by introducing variables that can have a degree of truth represented by values between 0 and 1. Fuzzy rules are if-then statements that define the relationship between fuzzy sets and the decisions made based on those sets. The fuzzy inference system evaluates these rules to make decisions [25]. Fuzzy logic handles uncertainty and imprecision effectively, allowing NPCs to make more nuanced decisions. Integrating fuzzy logic into Unreal Engine or Unity created NPCs significantly improves the fidelity and adaptability of driving simulations in total. Fuzzy controllers are designed to manage various aspects of the driving simulation, such as vehicle dynamics and driver behavior. Variables such as speed, distance, and steering angle are defined as inputs to the fuzzy controller, while acceleration, braking force, and steering adjustments are defined as outputs. A set of fuzzy rules must also be created to define the relationship between input and output variables. The following example demonstrates a creation of simple fuzzy logic controller visual script for controlling NPC vehicle speed based on distance to an obstacle using Unreal Engine's Blueprint system:

```
- Event Tick
    - Get Distance to Obstacle
    - Call EvaluateFuzzyLogic
        - Input: Distance to Obstacle
        - Output: Throttle
    - Set Throttle on Vehicle
```

For basic control, we can use C++ to implement the fuzzy logic system. First, we create the membership functions to handle distance evaluation:

```
float MembershipFunction(float x, float a, float b, float c)
{    if (x <= a || x >= c) return 0.0f;
     if (x < b) return (x - a) / (b - a);
     return (c - x) / (c - b);}
float Close(float distance)
{return MembershipFunction(distance,0.0f,0.0f,10.0f);}
float Medium(float distance)
{return MembershipFunction(distance,5.0f,10.0f,15.0f);}
float Far(float distance)
{return MembershipFunction(distance,10.0f,20.0f,30.0f);}
```

Next, we implement the fuzzy rules based on the input membership values:

```
float EvaluateThrottle(float distance) {
   float closeMembership = Close(distance);
   float mediumMembership = Medium(distance);
   float farMembership = Far(distance);

   // Fuzzy rules
   float throttleLow = closeMembership;
   float throttleMedium = mediumMembership;
   float throttleHigh = farMembership;

   // Weighted average for defuzzification
   float totalWeight = closeMembership +
    mediumMembership + farMembership;
   float throttle = (throttleLow * 0.3f + throttleMedium *
    0.6f + throttleHigh * 1.0f) / totalWeight;

   return throttle;}
```

Finally, we use the computed throttle value to control the vehicle in Unreal Engine:

```
void UpdateVehicleControl(float DeltaTime) {
   float distance = GetDistanceToObstacle();
   float throttle = EvaluateThrottle(distance);

   // Apply throttle to vehicle
   SetThrottle(throttle);}

// Called every frame
void Tick(float DeltaTime)
   {UpdateVehicleControl(DeltaTime);}
```

We can easily extend this by adding more complex rules, integrating with other vehicle dynamics parameters, and enhancing the membership functions to suit specific simulation requirements.

2.4. Interaction with Environment and Players

For NPCs in road traffic safety games, interaction with the game environment and the player is crucial for creating a realistic and educational experience. NPCs must be able to sense their surroundings, respond to dynamic changes, and interact with the player in meaningful ways. Perception systems allow NPCs to detect and interpret information from the game environment. Techniques such as Raycasting [26] and Sensor Fusion [27] enable NPCs to sense traffic signals, other vehicles, and obstacles. These systems provide the necessary data for NPCs to make informed decisions and navigate the environment safely. Flocking algorithms, such as the Boids algorithm [28], simulate the collective movement of groups of NPCs, such as pedestrian crowds or vehicle convoys. These algorithms help NPCs maintain cohesion, avoid collisions, and navigate crowded environments realistically. Flocking behavior is essential for simulating realistic traffic scenarios where multiple NPCs must interact and move together. Predictive collision avoidance techniques enable NPCs to anticipate and avoid potential collisions. By calculating the future positions of other vehicles and pedestrians, NPCs can adjust their paths to prevent accidents. These techniques are critical for creating realistic and safe traffic scenarios in road traffic safety games. Safety Driving Simulator® is a road traffic safety game that uses AI to create realistic traffic scenarios. The game employs the A* algorithm for pathfinding, FSMs for decision-making, and rule-based systems to enforce traffic regulations. NPCs in the game adapt their behavior based on player actions, providing a dynamic and educational learning experience that emphasizes safe driving practices (Fig. 5).

Figure 5. *Safety Driving Simulator GUI*

2.5. Machine learning

Machine learning techniques introduced a higher level of adaptability and intelligence to NPC control. By training NPCs on large datasets of player interactions, machine learning models can predict player behavior, optimize NPC strategies, and continuously improve their performance over time. This results in NPCs that can learn and adapt, providing a more personalized and engaging experience [29]. In *supervised learning*, the model is trained on labeled data, meaning the input data is paired with the correct output. The goal is to learn a mapping from inputs to outputs that can be applied to new, unseen data. For NPCs in traffic safety games, supervised learning can be used to model specific driving behaviors based on historical traffic data. Common algorithms include decision trees, support vector machines, and neural networks. *Unsupervised learning* deals with finding patterns in data without labeled outputs. Techniques include clustering (e.g., k-means) and dimensionality reduction (e.g., principal component analysis). Clustering algorithms can group NPC

behaviors based on similarities, allowing the creation of different behavior profiles without predefined labels. This approach is used to cluster driving behaviors and identify typical traffic scenarios that NPCs should be able to handle.

The first step in applying machine learning to NPC design is collecting and preparing relevant data. This can include player interactions, game states, and desired NPC behaviors. Once data is prepared, the next step is to train and validate machine learning models. This involves selecting appropriate algorithms and tuning their parameters to optimize performance. For example, a neural network can be trained using backpropagation to learn complex behaviors from the dataset. The network adjusts its weights based on the error between predicted and actual outcomes. After training, the machine learning models are integrated into the game's NPC control systems, enabling real-time decision-making and adaptation.

Reinforcement learning involves training agents to make sequences of decisions by rewarding desired behaviors. Agents learn to maximize cumulative rewards through trial and error [30]. Q-learning is a reinforcement learning algorithm where an NPC learns a policy to take actions that maximize its expected future rewards, adjusting its strategy based on received rewards and penalties. In the context of traffic simulations, reinforcement learning has been used to optimize traffic signal control, model driver behavior, and develop intelligent transportation systems.

The first step in applying RL to NPC design is defining the environment. This includes specifying the road layout, traffic rules, and the types of vehicles and obstacles that the NPCs will encounter. Several environment components must be defined: *state space* (current situation in the traffic simulation, including the positions and velocities of vehicles, traffic light statuses, and road conditions), *action space* (set of actions available to NPCs, such as acceleration, braking, lane changing, and turning), and *reward function* (feedback to the NPCs based on their actions, encouraging safe and efficient driving behaviors). The following Python code can be used for defining the traffic environment:

```
import numpy as np
class TrafficEnvironment:
    def __init__(self):
        self.state = None
        self.reset()
    def reset(self):
        self.state = np.array([0, 0]) # Init state:[pos, vel]
        return self.state
    def step(self, action):
        position, velocity = self.state
# Apply action: 0 = accel, 1 = brake, 2 = maintain speed
        if action == 0:
            velocity += 1
        elif action == 1:
            velocity -= 1
# Update position
        position += velocity
# Define a simple reward function to encourage
  maintaining a velocity of 10
        reward = -abs(velocity - 10)
# Update state
        self.state = np.array([position, velocity])
# Check if simulation is done
        done = position < 0 or position > 100
        return self.state, reward, done
    def render(self):
        print(f"Position: {self.state[0]}, Velocity:
        {self.state[1]}")
# Example of using the environment
env = TrafficEnvironment()
state = env.reset()
print(f"Initial state: {state}")
# Example action: accelerate
action = 0
next_state, reward, done = env.step(action)
print(f"Next state: {next_state}, Reward: {reward},
  Done: {done}")
```

The NPCs are treated as reinforcement learning agents that learn to navigate the environment. The agent's goal is to maximize the cumulative reward by making optimal driving decisions. A simple Q-learning algorithm without any external libraries will be used to keep it straightforward and understandable.:

```
class QLearningAgent:
    def __init__(self, num_states, num_actions,
     alpha=0.1, gamma=0.9, epsilon=0.1):
        self.num_states = num_states
        self.num_actions = num_actions
        self.alpha = alpha  # Learning rate
        self.gamma = gamma  # Discount factor
        self.epsilon = epsilon  # Exploration rate
        self.q_table = np.zeros((num_states,
          num_actions))  # Q-value table
    # Explore or exploit
    def choose_action(self, state):
        if np.random.rand() < self.epsilon:
            return np.random.choice(self.num_actions)
        else:
            return np.argmax(self.q_table[state])
    def update_q_table(self, state, action, reward,
       next_state):
        best_next_action =
          np.argmax(self.q_table[next_state])
        td_target = reward + self.gamma *
          self.q_table[next_state, best_next_action]
        self.q_table[state, action] += self.alpha *
          (td_target - self.q_table[state, action])
# Define states and actions
num_states = 10  # Example number of discrete states
num_actions = 3  # Accelerate, brake, maintain speed
# Create Q-learning agent
agent = QLearningAgent(num_states, num_actions)
# Training example
for episode in range(100):
    state = env.reset()
    state = int(state[1])  # Use velocity as simple state
    for step in range(50):
        action = agent.choose_action(state)
        next_state, reward, done = env.step(action)
```

```
        next_state = int(next_state[1])  # Use velocity as
                                           next state
        agent.update_q_table(state, action, reward,
          next_state)
        state = next_state
        if done:
            break
print(f"Q-table after training:\n{agent.q_table}")
```

The training process involves the agent interacting with the environment, collecting experiences, and updating its policy based on the received rewards. The following code implements reinforcement learning based NPCs:

```
class NPC:
    def __init__(self, agent, environment):
        self.agent = agent
        self.environment = environment
        self.state = self.environment.reset()
    def act(self):
        state = int(self.state[1])  # Use velocity as state
        action = self.agent.choose_action(state)
        self.state, reward, done =
          self.environment.step(action)
        return self.state, reward, done
# Instantiate NPC with trained agent
npc = NPC(agent, env)
# Simulate NPC behavior in the game
for step in range(100):
    state, reward, done = npc.act()
    env.render()
    if done:
        break
```

Reinforcement learning provides a powerful framework for designing adaptive and realistic NPCs in road traffic safety games. By leveraging various techniques, developers can create NPCs that exhibit complex and human-like driving behaviors.

2.6. Deep learning

Deep learning involves the use of artificial neural networks (ANNs) with multiple hidden layers (hence "deep") to model and learn complex relationships within data. Deep learning has revolutionized various fields, including computer vision, natural language processing, and autonomous driving, by enabling systems to learn complex patterns from large datasets. In the context of NPC design for traffic simulations, deep learning can be used to model and predict driving behaviors, making NPCs more adaptive and realistic. By leveraging neural networks with multiple layers, deep learning models can learn complex patterns and behaviors from large datasets, enabling NPCs to exhibit more sophisticated and realistic interactions [31]. Deep neural networks (DNNs) consist of multiple layers of neurons that process input data to produce an output. DNNs can model complex relationships in data, making them suitable for tasks such as behavior prediction and decision-making in NPCs. Convolutional Neural Networks (CNNs) can be employed to enable NPCs to recognize and interpret visual data from the game environment, such as identifying objects, obstacles, scene understanding, and navigation [32]. CNNs are specialized for processing grid-like data structures, such as images. In traffic simulations, CNNs can be used to analyze visual inputs, such as traffic camera feeds, to detect and respond to various traffic scenarios. CNNs can process visual data in real time, allowing NPCs to respond immediately to changes in the game environment. An NPC using a CNN can perform detailed scene analysis, identifying multiple objects and their relationships within the environment, leading to more informed decision-making.

Recurrent Neural Networks (RNNs) and *Long Short-Term Memory* (LSTM) networks are designed to handle sequential data, making them suitable for tasks requiring memory of past events [33]. In NPC control, these networks enable characters to remember and learn from previous interactions, leading to more coherent and contextually appropriate behaviors. An NPC can use an RNN to analyze sequences of player actions and predict future actions, adjusting its strategy accordingly to create more challenging gameplay.

Reinforcement Learning (RL) involves training an AI agent to make decisions by rewarding desired actions and penalizing undesired ones. It is a type of machine learning where agents learn to make decisions by taking actions in an environment to maximize cumulative rewards. RL involves an agent, environment, actions, states, and rewards. Training NPCs involves defining the states, actions, and rewards within the game environment and iteratively improving the NPC's policy through exploration and exploitation. Designing an appropriate reward function is crucial for guiding the NPC toward desired behaviors. Creating a simulation environment that mimics the actual game is essential for training NPCs without the constraints of real-time gameplay. This allows for accelerated learning and extensive experimentation.

Combining deep learning with reinforcement learning (*deep reinforcement learning*) allows NPCs to learn complex strategies by interacting with the environment and maximizing cumulative rewards. Deep reinforcement learning can be used to train NPCs to develop advanced strategies, such as planning and adapting tactics based on player actions and game state changes. When combined with deep learning, RL can be used to train NPCs to navigate complex traffic environments by learning from interactions with the environment. Deep reinforcement learning models require substantial amounts of data for training. This involves collecting relevant gameplay data and preprocessing it to ensure it is suitable for neural network training. Data might include player actions, NPC responses, environmental conditions, and game outcomes. Preprocessing steps can

involve normalization, augmentation, and segmentation of this data. Training learning models involves feeding the preprocessed data into the neural network, adjusting the weights through backpropagation, and validating the model to prevent overfitting. Once trained, deep reinforcement learning models are integrated into the game's NPC control systems for real-time decision-making and behavior generation. This model excels at processing complex and high-dimensional data, enabling NPCs to understand and react to intricate game environments and scenarios. This adaptability results in more challenging and unpredictable gameplay, as NPCs can dynamically modify their strategies and tactics.

3. ETHICAL CHALLENGES IN NPC DESIGN FOR ROAD TRAFFIC SAFETY GAMES

Road traffic safety games serve as critical tools for driver training, education, and research. The specificity of NPCs used in these games are that they simulate realistic traffic behaviors, providing learners with scenarios to practice driving skills and understand traffic dynamics, which are crucial and critical elements in real-world traffic environment movement and orientation. AI techniques, particularly machine learning and deep learning, enable NPCs to exhibit complex behaviors and adapt to diverse traffic conditions, enhancing the effectiveness of these informal and non-formal educational tools. While AI-driven NPC design offers numerous benefits, it also introduces ethical challenges that must be addressed:

- *Safety* and *reliability* by ensuring that AI-driven NPCs make safe and reliable decisions in all traffic scenarios, especially in dynamic and unpredictable environments. The NPC algorithms must be extensively tested and validated in order to verify their appropriate behavior across a wide range of scenarios and realistic situations [34];
- *Fairness* by avoiding bias in NPC behaviors that may disproportionately affect certain groups or communities. Biased behaviors can be efficiently suppressed and prevented by including diverse demographic groups and traffic scenarios into training datasets. In addition, various techniques can be implemented to detect and mitigate biases in AI models during its development and deployment.
- *Transparency* of how NPCs make decisions to build trust among users and stakeholders. Explainable AI (XAI) [35] should be utilized to provide users with insights into the decision-making process of NPCs in combination with user interface design that effectively communicate NPC behaviors;
- *Accountability* for establishing mechanisms to attribute responsibility for NPC actions and decisions, particularly in the event of accidents or incidents. Legal and regulatory frameworks that outline responsibilities and liabilities for AI-driven NPCs in gaming environments should be integrated in combination with clear ethical guidelines [36];
- *Societal impact* and *societal implications* of AI-controlled NPCs in shaping player attitudes and behaviors towards road safety. Comprehensive assessments and evaluation of potential ethical and/or societal consequences of AI-driven NPCs should be conducted in combination with raising awareness about the role and impact of AI in road traffic safety games [37];
- *Data privacy* concerns arise from the collection, storage, and utilization of personal information in AI-driven NPC design. User consent regarding the collection and use of their data for NPC design purposes should be obtained. Data anonymization or pseudonymization techniques should also be implemented to protect individual identities.

4. CONCLUSION

AI-driven NPCs contribute to a more immersive gaming experience by exhibiting lifelike behaviors and interactions. This realism is crucial for maintaining player engagement and creating believable game worlds. NPCs that can react dynamically to the player and the environment make the game world feel alive and responsive. AI models add layers of strategic depth to games. In strategy and tactical games, intelligent NPCs can challenge players with sophisticated tactics, requiring players to think critically and adapt their strategies. This complexity enhances replayability and keeps players engaged over longer periods. Adaptive AI allows NPCs to tailor their behavior to individual player styles and preferences. This personalization ensures that games remain challenging and enjoyable, catering to both novice and experienced players. By analyzing player behavior and adjusting NPC strategies, AI creates a more dynamic and satisfying gaming experience.

Advances in deep learning, natural language processing, and AI ethics will further enhance NPC capabilities, enabling more sophisticated interactions. Future research should focus on improving AI efficiency, reducing development costs, and exploring new ways to integrate AI seamlessly into game design.

Implementing advanced AI techniques in NPC control is computationally intensive, and can potentially affect game performance. Balancing the complexity of AI algorithms with the available computational resources is a critical challenge for developers. Deep learning models require significant computational resources, which can be a limiting factor for real-time applications in games.

Additionally, training deep learning models necessitates large datasets that can be difficult to obtain and curate. Ensuring the quality and diversity of training data is essential for developing robust and generalizable NPC behaviors. The use of AI models raises ethical questions regarding the behavior and representation of NPCs. Ensuring that NPC behavior aligns with ethical standards and does not perpetuate harmful stereotypes or behaviors is essential.

The future of using AI in NPC control is promising, with potential advancements in unsupervised learning, transfer learning, and explainable AI. These techniques will further enhance NPC intelligence, making characters more autonomous and capable of even richer interactions. While challenges remain, the continued evolution of AI integration promises even greater innovations in NPC control, pushing the boundaries of what is possible in digital game design. As AI continues to evolve, it is essential to ensure that it complements and enhances the creative aspects of game design rather than overshadowing them.

ACKNOWLEDGEMENTS

This study was supported by the Ministry of Science, Technological Development and Innovation of the Republic of Serbia, and these results are parts of the Grant No. 451-03-66/2024-03/200132 with University of Kragujevac, Faculty of Technical Sciences Čačak.

REFERENCES

[1] Schijven, M. P., & Kikkawa, T. (2022). Is there any (artificial) intelligence in gaming? *Simulation & Gaming, 53*(4), 315–316. doi:10.1177/10468781221101685

[2] Aleksić, V., Ivanović, M. (2017). Early Adolescent Gender and Multiple Intelligences Profiles as Predictors of Digital Gameplay Preferences. *Croatian Journal of Education, 19*(3), pp. 697-727. doi:10.15516/cje.v19i3.2262

[3] Vermesan, O. (2021). Artificial Intelligence for Digitising Industry. *Artificial Intelligence for Digitising Industry*, 1–541. doi:10.13052/rp-9788770226639

[4] Aleksić, V. (2023). *Razvoj digitalnih igara*. Čačak: Fakultet tehničkih nauka. ISBN 978-86-7776-269-8

[5] Roberts, P. (2022). *Artificial Intelligence in Games*. CRC Press. doi:10.1201/9781003305835

[6] Millington, I. (2019). Procedural Content Generation. *AI for Games*, 669–738. doi:10.1201/9781351053303-8

[7] Sun, L., Kangas, M., & Ruokamo, H. (2023). Game-based features in intelligent game-based learning environments: a systematic literature review. *Interactive Learning Environments*, 1–17. doi:10.1080/10494820.2023.2179638

[8] Mendoza Guevarra, E. T. (2020). *Creating Game Environments in Blender 3D*. Apress. doi:10.1007/978-1-4842-6174-3

[9] Johnson, G. (2019). Non-Player Characters, Foes, and Monsters. Developing Creative Content for Games, 151–168. doi:10.1201/9781315152554-16

[10] Wagner, F., Schmuki, R., Wagner, T., & Wolstenholme, P. (2006). *Modeling software with finite state machines: a practical approach*. Auerbach Publications. ISBN 978-08-4938-086-0

[11] Lilis, Y., & Savidis, A. (2014). An Integrated Development Framework for Tabletop Computer Games. *Computers in Entertainment, 12*(3), 1–34. doi:10.1145/2702109.2633423

[12] Filho, M. E. M., Souza, A. J. S., Tedesco, P. C. A. R., Silva, D. R. D., & Ramalho, G. L. (2009). An Integrated Development Model for Character-Based Games. *2009 VIII Brazilian Symposium on Games and Digital Entertainment*. doi:10.1109/sbgames.2009.30

[13] Jung, W.-J. (2022). RPG User Play Observation Learning-based Guide NPC AI Reinforcement Learning. *Journal of Korea Game Society, 22*(5), 73–83. doi:10.7583/jkgs.2022.22.5.73

[14] Kay, M., & Powley, E. J. (2018). The effect of visualising NPC pathfinding on player exploration. *Proceedings of the 13th International Conference on the Foundations of Digital Games*. doi:10.1145/3235765.3235824

[15] Handy Permana, S. D., Yogha Bintoro, K. B., Arifitama, B., & Syahputra, A. (2018). Comparative Analysis of Pathfinding Algorithms A *, Dijkstra, and BFS on Maze Runner Game. *IJISTECH (International Journal Of Information System & Technology)*, 1(2), 1. doi:10.30645/ijistech.v1i2.7

[16] Krisdiawan, R. A., Permana, A., Darmawan, E., Nugraha, F., & Kriswandiyanto, A. (2021). Implementation Dijkstra's Algorithm for Non-Players Characters in the Game Dark Lumber. *Journal of Physics: Conference Series, 1933*(1), 012006. doi:10.1088/1742-6596/1933/1/012006

[17] Sazaki, Y., Satria, H., & Syahroyni, M. (2017). Comparison of A* and dynamic pathfinding algorithm with dynamic pathfinding algorithm for NPC on car racing game. *2017 11th International Conference on Telecommunication Systems Services and Applications (TSSA)*. doi:10.1109/tssa.2017.8272918

[18] Saranya, C., Unnikrishnan, M., Ali, S. A., Sheela, D. S., & Lalithambika, Dr. V. R. (2016). Terrain Based D∗ Algorithm for Path Planning. *IFAC-PapersOnLine, 49*(1), 178–182. doi:10.1016/j.ifacol.2016.03.049

[19] Daud, A. A. G., Muhaqiqin, M., & Sintaro, S. (2021). Comparison of Jump Point Search Algorithms and Basic Theta* Algorithms in Determining the Shortest Route in NPC in Maze Games. In *The 1st International Conference on Advanced Information Technology and*

Communication (IC-AITC). 3-4 September, Universitas Teknokrat Indonesia.

[20]Pandey, G., Rao, K. R., & Mohan, D. (2014). A Review of Cellular Automata Model for Heterogeneous Traffic Conditions. *Traffic and Granular Flow '13*, 471-478. doi:10.1007/978-3-319-10629-8_52

[21]Zhang, X., & Zhang, X. (2022). Based on Navmesh to implement AI intelligent pathfinding in three-dimensional maps in UE4. *Proceedings of the 2022 5th International Conference on Algorithms, Computing and Artificial Intelligence*, 23-25th Dec, Sanya, China, pp.1-5. doi:10.1145/3579654.3579752

[22]Kapi, A. Y. (2020). A Review on Informed Search Algorithms for Video Games Pathfinding. *International Journal of Advanced Trends in Computer Science and Engineering, 9*(3), 2756–2764. doi:10.30534/ijatcse/2020/42932020

[23]Cho, D.-H., Lee, Y.-H., Kim, J.-H., Park, S.-Y., & Rhee, D.-W. (2011). NPC Control Model for Defense in Soccer Game Applying the Decision Tree Learning Algorithm. *Journal of Korea Game Society, 11*(6), 61–70. doi:10.7583/jkgs.2011.11.6.61

[24]Belle, S., Gittens, C., & Graham, T. C. N. (2019). Programming with Affect: How Behaviour Trees and a Lightweight Cognitive Architecture Enable the Development of Non-Player Characters with Emotions. *2019 IEEE Games, Entertainment, Media Conference (GEM).* 18-21th June, New Haven, USA. doi:10.1109/gem.2019.8811542

[25]Hubble, A., Moorin, J., & Khuman, A. S. (2021). Artificial Intelligence in FPS Games: NPC Difficulty Effects on Gameplay. *Fuzzy Logic*, 165–190. doi:10.1007/978-3-030-66474-9_11

[26]Ying, Z., Edwards, N., & Kutuzov, M. (2024). Efficient Visibility Approximation for Game AI using Neural Omnidirectional Distance Fields. *Proceedings of the ACM on Computer Graphics and Interactive Techniques, 7*(1), 1–15. doi:10.1145/3651289

[27]Choi, H., Han, S., Jeon, J., Ahn, S., & Yoo, J. (2024). Simulation-Based SOTIF Hazard Analysis and Risk Assessment Methodology for Autonomous Driving System. *Transaction of the Korean Society of Automotive Engineers, 32*(4), 331–347. doi:10.7467/ksae.2024.32.4.331

[28]Knievel, C., Pejic, A., Krüger, L., Ziegler, C., & Adamy, J. (2023). Boids Flocking Algorithm for Situation Assessment of Driver Assistance Systems. *IEEE Open Journal of Intelligent Transportation Systems, 4*, 71–82. doi:10.1109/ojits.2023.3236985

[29]Edwards, G., Subianto, N., Englund, D., Goh, J. W., Coughran, N., Milton, Z., Mirnateghi, N., & Ali Shah, S. A. (2021). The Role of Machine Learning in Game Development Domain - A Review of Current Trends and Future Directions. *2021 Digital Image Computing: Techniques and Applications (DICTA)*, pp. 1-7, 29-30th November, Gold Coast, Australia. doi:10.1109/dicta52665.2021.9647261

[30]Cherukuri, A., & Glavin, F. G. (2022). Balancing the Performance of a FightingICE Agent using Reinforcement Learning and Skilled Experience Catalogue. *2022 IEEE Games, Entertainment, Media Conference (GEM)*, pp.1-6, 27-30th November, St. Michael, Barbados. doi:10.1109/gem56474.2022.10017566

[31]Maulana, A., Mardi, S., Yuniarno, E. M., & Suprapto, Y. K. (2022). Behavior NPC Prediction Using Deep Learning. *2022 International Conference on Computer Engineering, Network, and Intelligent Multimedia (CENIM)*, pp. 1-5, 22-23th November, Surabaya, Indonesia. doi:10.1109/cenim56801.2022.10037328

[32]Sehrawat, A., & Raj, G. (2018). Intelligent PC Games: Comparison of Neural Network Based AI against Pre-Scripted AI. *2018 International Conference on Advances in Computing and Communication Engineering (ICACCE),* pp. 378-383, 22-23th June, Paris, France. doi:10.1109/icacce.2018.8441745

[33]Ryan, J., Summerville, A. J., Mateas, M., & Wardrip-Fruin, N. (2016). Translating Player Dialogue into Meaning Representations Using LSTMs. *Lecture Notes in Computer Science*, 383–386 doi:10.1007/978-3-319-47665-0_38

[34]Wang, J., & Zhao, R. (2022). Deep reinforcement learning and application in self-driving. *Theories and Practices of Self-Driving Vehicles*, 307–326. doi:10.1016/b978-0-323-99448-4.00010-2

[35]Arun Sampaul Thomas, G., Muthukaruppasamy, S., Nandha Gopal, J., Sudha, G., & Saravanan, K. (2024). Unleashing the Power of XAI (Explainable Artificial Intelligence). *Explainable AI (XAI) for Sustainable Development*, 303–316. doi:10.1201/9781003457176-18

[36]Vingilis, E., Yıldırım-Yenier, Z., Fischer, P., Wiesenthal, D. L., Wickens, C. M., Mann, R. E., & Seeley, J. (2016). Self-concept as a risky driver: Mediating the relationship between racing video games and on-road driving violations in a community-based sample. *Transportation Research Part F: Traffic Psychology and Behaviour, 43*, 15–23. doi:10.1016/j.trf.2016.09.021

[37]Forneris, L., Pighetti, A., Lazzaroni, L., Bellotti, F., Capello, A., Cossu, M., & Berta, R. (2023). Implementing Deep Reinforcement Learning (DRL)-based Driving Styles for Non-Player Vehicles. *International Journal of Serious Games, 10*(4), 153–170. doi:10.17083/ijsg.v10i4.638