AKADEMIA GÓRNICZO-HUTNICZA IM. STANISŁAWA STASZICA W KRAKOWIE

WYDZIAŁ ELEKTROTECHNIKI, AUTOMATYKI, INFORMATYKI I INŻYNIERII BIOMEDYCZNEJ

Praca dyplomowa

Optymalizacja pracy windy z wykorzystaniem algorytmów sztucznej inteligencji

Elevator behaviour optimization using artificial intelligence algorithms

Autor: *Mateusz Wojtulewicz*
Kierunek studiów: *Informatyka*
Opiekun pracy: *prof. dr hab. inż. Tomasz Szmuc*

Kraków, 2022

Serdecznie dziękuję Panu Promotorowi prof. dr hab. inż. Tomaszowi Szmucowi za zaangażowanie i nieocenioną pomoc przy tworzeniu pracy dyplomowej.

Dziękuję również firmie ASTOR *sp. z o.o. za udostępnienie historycznych odczytów systemu windy z inteligentnego budynku swojej siedziby.*

Spis treści

1. Wprowadzenie

Windy są powszechnie stosowane w komunikacji między piętrami. Zainstalowane w niemal każdym budynku przewożą dziennie setki pasażerów. Według badań *IBM* z roku 2010 pracownicy w Nowym Yorku na przestrzeni 12 miesięcy spędzili sumarycznie czas 16 lat czekając na przyjazd windy oraz prawie 6 lat wewnątrz windy [7]. Czynność oczekiwania powoduje nie tylko frustrację ale również przeliczalne straty.

1.1. Motywacja

Zadanie sterowania ruchu windą w sposób minimalizujący czas oczekiwania jej użytkowników w ustalonym i deterministycznym środowisku to problem udowodniony jako $\mathcal{NP}$-trudny [14]. Dodatkowo, w systemie rzeczywistym pojawia się niepewność, wymagająca reagowania bez posiadania pełnej wiedzy o przyszłych wydarzeniach. Z tego względu powstało wiele algorytmów heurystycznych adresujących opisane zagadnienie. Również rozwiązania oparte na algorytmach sztucznej inteligencji zostały częściowo przebadane.

Usprawnienie strategii sterowania windą, potencjał drzemiący w technikach uczenia ze wzmocnieniem oraz zwiększone zainteresowanie tymi technikami były motywacją do podjęcia tego tematu. Efektywne sterowanie jest przedmiotem szerokiego zainteresowania użytkowników, a proces intuicyjnie oczywisty. Z drugiej strony wyznaczenie algorytmu optymalizującego jest (jak wspomniano wyżej) problemem trudnym. Dostarcza to szerokich możliwości do badania uzyskania poprawy przez zastosowanie algorytmów AI, a w szczególności zaawansowanych technik uczenia ze wzmocnieniem.

1.2. Cel pracy

Celem niniejszej pracy jest ocena przydatności algorytmów uczenia ze wzmocnieniem do rozwiązania problemu optymalnego sterowania windą. Badania przeprowadzono przez porównanie wypracowanych strategii z istniejącymi rozwiązaniami heurystycznymi. Porównanie wykonano z zastosowaniem zaproponowanych metryk. Na potrzeby pracy został zaimplementowany uniwersalny symulator windy. Opracowano i zamodelowano rozkład ruchu w rzeczywistym budynku dzięki odczytom z systemu sterowania w inteligentnym budynku firmy ASTOR. Dodatkowym rezultatem badań jest możliwość zestawienia i analizy przebiegu nauki zaawansowanych algorytmów uczenia ze wzmocnieniem w ujednoliconym środowisku problemu.

1.3. Zawartość pracy

W rozdziale 2 zdefiniowano pojęcie sztucznej inteligencji. Przedstawiono także jej podział na obszary względem poziomu zaawansowania oraz względem techniki rozwiązywania problemu.

Rozdział 3 rozwija paradygmat techniki uczenia ze wzmocnieniem opisując jej założenia i strukturę. Wyszczególnia i opisuje algorytmy i zaawansowane techniki, które zostały wykorzystane w pracy.

Zadaniem rozdziału 4 jest scharakteryzowanie problemu decyzyjnego optymalnej obsługi systemu windy. Przedstawiono w nim jego własności oraz opisano istniejące heurystyczne strategie kontroli. Na końcu wyliczono i scharakteryzowano wybrane nowoczesne rozwiązania tego problemu.

Rozdział 5 przedstawia metodykę przyjętą podczas pracy oraz elementy projektu rozwiązania. Przedstawia strukturę stworzonego symulatora i opisuje proces modelowania rzeczywistego rozkładu ruchu na postawie odczytów z budynku ASTOR. Opisuje również strukturę konstrukcji środowisk decyzyjnych oraz algorytmów nauki.

W rozdziale 6 przedstawiono i opisano architekturę programu, oraz wyszczególniono wykorzystane technologie.

W rozdziale 7 wymieniono i opisano wybrane pary środowiska i działającego w nim agenta uczenia ze wzmocnieniem, które opracowano w trakcie pracy i zasługują na uwagę. Porównano przebieg procesów nauki agentów wykorzystujących różne algorytmy i techniki.

W ramach rozdziału 8 wyjaśniono przebiegi symulacji testowych przeprowadzanych dla wytrenowanych agentów. W dalszej jego części porównano rezultaty agentów względem uzyskanych dla klasycznych algorytmów heurystycznych.

Ostatni rozdział 9 zawiera dyskusję nad wynikami pracy zakończoną oceniającymi wnioskami. Przedstawia również możliwe kierunki rozwoju opracowanych w pracy rozwiązań i narzędzi.

W celu zwiększenia czytelności i dla ułatwienia śledzenia głównego nurtu pracy przyjęto konwencję umieszczania w opisach pod tabelami i rysunkami wniosków bezpośrednio się do nich odnoszących. W ten sposób Czytelnik nie jest zmuszony przeskakiwać między tekstem a niekiedy odległym rysunkiem. W niektórych przypadkach generuje to jednak opisy o niestandardowych długościach.

2. Metody sztucznej inteligencji

W tym rozdziale opisano pojęcie sztucznej inteligencji oraz wymieniono jej obszary. Skupiono się także na podziale technik uczenia maszynowego, co stanowi wstęp teoretyczny do wykorzystanych w projekcie metod.

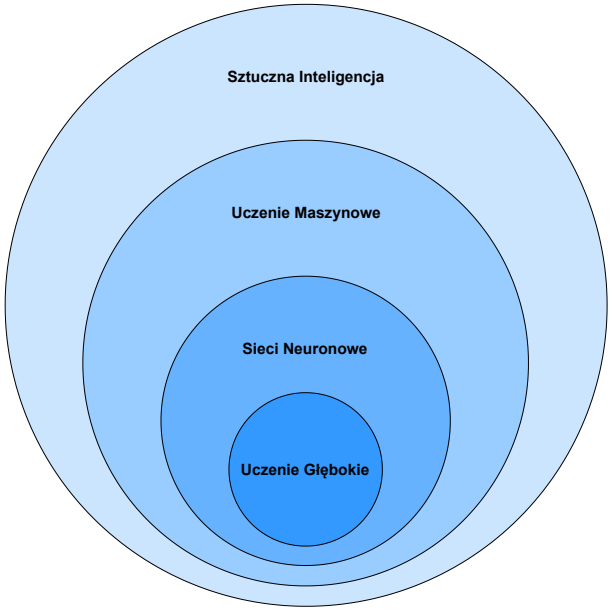
2.1. Charakterystyka

Pojęcie sztucznej inteligencji (ang. *Artificial Intelligence, AI*) zostało zaproponowane w 1956 roku przez amerykańskiego profesora Johna McCarthy'ego. Obecnie sztuczna inteligencja odnosi się do ogółu technik tworzenia systemów i programów komputerowych mających na celu naśladowanie inteligencji naturalnej, zwłaszcza ludzkiej. System taki zdolny jest obserwować swoje środowisko, analizować je i reagować w sposób maksymalizujący szansę na osiągnięcie zadanego celu. Spełnienie tego wymaga od algorytmu pewnego stopnia myślenia i działania racjonalnego, co jest bardzo trudne do kwantyzacji [20].

Tak sformułowana definicja obejmuje oczywiście dość szerokie spektrum algorytmów. Każdy bowiem program, który optymalizuje pewien proces decyzyjny, wspomaga go lub modeluje można zaliczyć do AI. Dlatego zawęża się go wyszczególniając kolejne, coraz bardziej zaawansowane obszary sztucznej inteligencji. Ich wzajemną zależność ilustruje rysunek 2.1.

2.2. Kręgi sztucznej inteligencji

Najszerszy z podzbiorów dziedziny sztucznej inteligencji, a więc całe jej spektrum, zawiera algorytmy i systemy spełniające wymienione wcześniej założenie naśladowania inteligencji naturalnej w rozwiązywaniu zadanego problemu. Każdy kolejny krąg specyfikuje wykorzystanie dodatkowych schematów, mających zazwyczaj swoje podłoże w matematyce lub statystyce, do zapewnienia lepszych wyników.



Rys. 2.1. Obszary sztucznej inteligencji

Źródło: opracowanie własne

2.2.1. Uczenie maszynowe

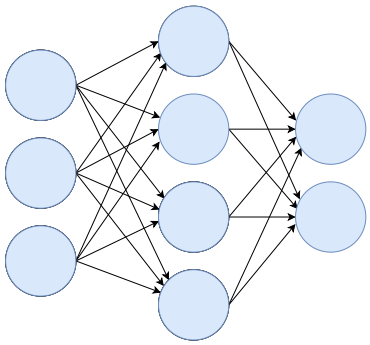
Pierwszą poddziedziną sztucznej inteligencji jest uczenie maszynowe (ang. *Machine Learning, ML*). Zakłada ono wykorzystanie doświadczenia do zwiększenia trafności podejmowania kolejnych decyzji. Na podstawie danych historycznych (zwanych również treningowymi) algorytm buduje model problemu poprzez znajdowanie wzorców w nich występujących. Poprzez użycie technik matematyczno-statystycznych poprawia się w rozwiązywaniu zadanego problemu i przybliża się do optymalnej strategii działania.

Wyszczególnia się trzy główne techniki uczenia maszynowego: uczenie nadzorowane, uczenie nienadzorowane oraz uczenie ze wzmocnieniem. Dokładniejszy opis zamieszczono w podrozdziale 2.3.

2.2.2. Sieci neuronowe

Sztuczne sieci neuronowe (ang. *Artificial Neural Networks, ANNs*), zwane zwykle po prostu sieciami neuronowymi (ang. *Neural Networks, NNs*) to systemy obliczeniowe zainspirowane strukturą sieci tworzoną przez neurony w mózgu zwierząt. Składają się z połączonych węzłów, zwanych sztucznymi neuronami, które modelują neurony w biologicznym mózgu. Każde połączenie, tak jak synapsy w mózgu biologicznym, zdolne jest przesyłać sygnał do innych neuronów. Sygnał to wartość rzeczywista, którą neuron po otrzymaniu przetwarza tzw. funkcją aktywacji (zwykle nieliniową) i przesyła dalej do wszystkich połączonych z nim neuronów. Sygnał przepływając przez połączenie między neuronami przemnażany jest przez skojarzoną

z danym połączeniem wagą, która w efekcie wzmacnia lub osłabia dany sygnał. Neurony są zwykle pogrupowane w warstwy. W typowej topologii sieci *feedforward* warstwy ułożone są sekwencyjnie od warstwy wejściowej, przez warstwy ukryte, po warstwę wyjściową. Ostatnia warstwa generuje sygnał wyjściowy sieci. Przykładowa struktura takiej sieci została przedstawiona na rysunku 2.2.



Rys. 2.2. Przykładowa struktura sieci *feedforward* składającej się z warstwy wejściowej (3 neurony), warstwy ukrytej (4 neurony) i warstwy wyjściowej (2 neurony). Strzałki ilustrują kierunek przepływu sygnału.

Źródło: opracowanie własne

Sygnał wejściowy, przepływając przez kolejne warstwy poddawany jest naprzemiennie liniowym i nieliniowym przekształceniom (te pierwsze w momencie sumowania wzmocnionych przez wagi sygnałów wchodzących do neuronu, drugie podczas nałożenia funkcji aktywacji). Dzięki temu sieć może modelować w przybliżeniu bardzo szerokie spektrum funkcji. Proces modelowania struktury sieci w celu osiągnięcia takiego stanu nazywa się treningiem.

Trening sieci neuronowych, jako że zawiera się w zbiorze technik uczenia maszynowego, korzysta z dostarczonych danych treningowych. Każdy przykład danych zawiera wejście i skojarzony z nim wzorcowy wynik. Sieć przetwarza dane wejściowe generując predykcję, która następnie porównywana jest z wzorcową wartością. Różnica między nimi to błąd sieci, który jest minimalizowany podczas nauki. Sieć w trakcie nauki zmienia wartości wag swoich połączeń zgodnie z zadanym algorytmem treningu. Najszerzej wykorzystywany algorytm stochastycznego spadku wzdłuż gradientu (ang. *Stochastic Gradient Descent, SGD*) wykonuje to poprzez obliczanie wartości gradientu funkcji błędu w przestrzeni wag, a następnie drobnej aktualizacji wag zgodnie z kierunkiem największego spadku gradientu. Takie zmiany przeprowadzane są wielokrotnie, co sprawia, że sieć sukcesywnie generuje coraz bliższe wyniki do wzorcowych.

2.2.3. Uczenie głębokie

Uczenie głębokie (ang. *Deep Learning, DL*) wykorzystuje sieci, które posiadają wiele warstw ukrytych. Taka struktura pozwala na ekstrakcję i hierarchizację bardziej skomplikowanych cech z nieprzetworzonych danych wejściowych. Dla porównania sieci niegłębokie wymagają wkładu człowieka celem przetworzenia danych treningowych na wejściowe, wyodrębnienia cech i ich wstępnej priorytetyzacji. Przykładowo w przetwarzaniu obrazów warstwy początkowe głębokiej sieci neuronowej mogą rozpoznawać proste cechy np. krawędzie, a dalsze warstwy identyfikować swoistości takie jak oczy, twarze czy symbole [3]. Dzięki temu uczenie głębokie znalazło swoje ogromne zastosowanie w obszarach wykorzystujących duże zbiory wysoko-abstrakcyjnych danych takich jak: wizja komputerowa (ang. *Computer Vision*), rozpoznawanie mowy czy przetwarzanie sygnałów przemysłowych.

2.3. Techniki uczenia maszynowego

W tej sekcji opisane zostały trzy najbardziej znaczące techniki uczenia maszynowego. Podział ten wynika z charakteru problemu, procesu nauki oraz celu algorytmu.

2.3.1. Uczenie nadzorowane

Technika uczenia nadzorowanego (ang. *Supervised Learning*) zakłada istnienie uprzednio opisanych par wejścia-wyjścia danych treningowych. Celem algorytmu jest interpolacja funkcji przekształcającej parametry wejściowe na wartość wyjściową. Wyodrębnia się dwie główne grupy problemów: klasyfikacja i regresja. W tej pierwszej wartością opisywaną jest określenie przynależności badanego elementu do jednej ze skończonego zbioru klas. W drugiej natomiast jest nią wartość z ciągłej dziedziny. Przykładem problemu klasyfikacji jest rozpoznawanie ręcznie narysowanej cyfry na podstawie jej obrazu. Znajdowanie zależności między wagą a wzrostem myszy to z kolei możliwy problem regresyjny.

2.3.2. Uczenie nienadzorowane

W problemach rozwiązywanych techniką uczenia nienadzorowanego (ang. *Unsupervised Learning*) algorytm nie ma dostępu do opisanych par treningowych ani metryk pozwalającej określić dokładność decyzji. Dlatego też w pierwszej fazie musi on odkryć wszelkie wzorce występujące naturalnie w dostarczonych danych. Najbardziej znanym algorytmem uczenia nienadzorowanego jest klasteryzacja, w której algorytm grupuje dane treningowe posiadające podobne cechy w zbiory zwane klastrami. Przykładem takiego zastosowania może być system dostarczania spersonalizowanych reklam poprzez modelowanie i grupowanie konsumentów.

2.3.3. Uczenie ze wzmocnieniem

Uczenie ze wzmocnieniem (ang. *Reinforcement Learning, RL*) to metoda polegająca na nauce poprzez zbieranie doświadczenia w interakcji z otoczeniem. Swoją inspirację technika *RL* czerpie z modelu nauki występującego w naturze, również u ludzi. Niemowlę bada i interpretuje otoczenie korzystając z wrodzonych umiejętności percepcji. Poprzez ciągłą interakcję odkrywa i kojarzy decyzje ze skutkami. W teorii dalsze wybory bazować będą na tych odkryciach. W podobny sposób agent *RL* uczy się maksymalizować szansę na zwycięstwo, np. w grach komputerowych takich jak: szachy, *League of Legends* czy *StarCraft II* [18].

Ponieważ opracowane w pracy rozwiązanie bazuje na technice uczenia ze wzmocnieniem poświęcony jej został cały kolejny rozdział 3.

3. Uczenie ze wzmocnieniem

W niniejszym rozdziale rozwinięty został paradygmat techniki uczenia przez wzmacnianie. Opisane zostały motywacje, struktura oraz przegląd algorytmów ją wykorzystujących. Przedstawione zostały również zaawansowane struktury i techniki poprawiające skuteczność i stabilność procesu nauki.

3.1. Koncepcja

Uczenie ze wzmocnieniem (ang. *Reinforcement Learning, RL*) jest obszarem *ML* skupiającym się na na uczeniu przez interakcję [16]. Bada sposoby tworzenia inteligentnych agentów zdolnych eksplorować swoje otoczenie i tym sposobem uczących się strategii reagowania na jej dynamikę w sposób maksymalizujący skumulowaną nagrodę.

Algorytmy *RL* imitują proces nauki występujący u zwierząt w naturze [10]. Przykładowo pies postrzega pewne sygnały, np. ból czy głód, jako wzmocnienie negatywne, a inne, takie jak radość czy pokarm, jako wzmocnienie pozytywne. Poprzez nieustanne reagowanie na zmieniające się środowisko będzie unikał działań, które mogą zadać mu ból, a wybierał te zwiększające szanse na otrzymanie jedzenia, na przykład przez dokładne wykonywanie poleceń swojego właściciela.

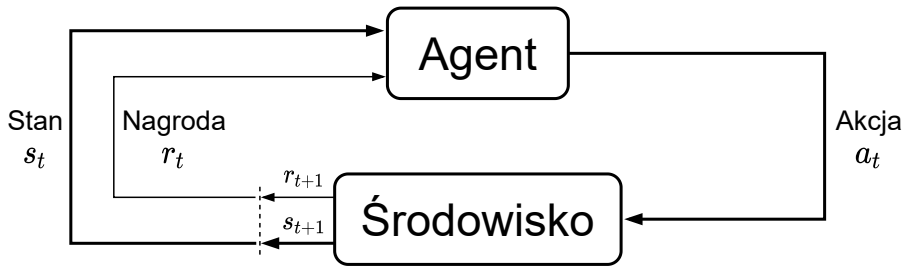
W odróżnieniu od uczenia nadzorowanego algorytm nie otrzymuje podczas nauki wzorcowych par wejścia-wyjścia. W kontekście problemów *RL* są one rozumiane jako optymalna akcja dla danego stanu środowiska. Poprzez interakcję ze środowiskiem zbiera natomiast doświadczenie konsekwencji podejmowanych działań. Jego zadaniem jest odkrycie, jakie akcje w danym stanie prowadzą do zdobycia największej nagrody. W rezultacie algorytmy te są wykorzystywane z powodzeniem w problemach, gdzie optymalna strategia działania nie jest znana. Swoje zastosowanie znajdują w dyscyplinach naukowych takich jak m.in.: Teoria Gier, Teoria Sterowania, Optymalizacja Procesów Dyskretnych, Optymalizacja Systemów Wieloagentowych.

3.1.1. Środowisko

Będące otoczeniem agenta środowisko może być procesem rzeczywistym lub modelem imitującym jego dynamikę. W pierwszym przypadku potrzebny jest sposób jego obserwacji i interpretacji w czasie rzeczywistym. W drugim modelem jest pewien proces decyzyjny Markova (ang. *Markov Decision Process, MDP*), na który składają się zbiór możliwych stanów S, zbiór dostępnych akcji A oraz dwie funkcje: przejścia $\mathrm{T}(s, a, s') : S \times A \times S \to \langle 0, 1 \rangle$ i nagrody $\mathrm{R}(s, a, s') : S \times A \times S \to \mathbb{R}$. Pierwsza określa dynamikę środowiska, czyli prawdopodobieństwo spowodowania zmiany środowiska ze stanu s na s' poprzez wykonanie akcji a. Druga odnosi się do otrzymywanej wówczas nagrody.

3.1.2. Interakcja

Agent wpływa na środowisko w czasie. W każdym dyskretnym kwancie czasu t, agent obserwuje stan środowiska s_t z przestrzeni stanów S i wybiera akcję a_t ze zbioru możliwych akcji A. W efekcie otrzymuje nagrodę $r_t \in \mathbb{R}$ i obserwuje nowy stan środowiska $s_{t+1} \in S$ zgodnie z dynamiką środowiska lub jej modelem. Opisany schemat, zwany pętlą interakcji, został przedstawiony na diagramie 3.1.



Rys. 3.1. Pętla interakcji agent-środowisko w uczeniu ze wzmocnieniem.

Źródło: opracowanie własne na podstawie Sutton i Barto [16]

3.1.3. Strategia i funkcje wartości

W większości algorytmów uczenia ze wzmocnieniem przeprowadza się estymację funkcji wartości. Określają one, w kontekście otrzymywanych nagród, jak korzystnie jest dla agenta znaleźć się w konkretnym stanie (lub wykonać daną akcję w tym stanie). Jest to oczywiście zależne od zachowania agenta, zwanego jego strategią. Strategię (ang. *policy*) definiuje się jako funkcję π modelującą wybory agenta:

$$\begin{aligned} &\pi : A \times S \to \langle 0, 1 \rangle \\ &\pi(a, s) = Pr(a_t = a \mid s_t = s) \end{aligned} \tag{3.1}$$

Strategia stochastyczna określa prawdopodobieństwo wybrania przez agenta akcji a obserwując stan s środowiska. Istnieją również strategie deterministyczne:

$$\begin{aligned} &\pi : S \to A \\ &\pi(s_t = s) = a \end{aligned} \tag{3.2}$$

Funkcja wartości stanu (ang. *state-value function*) $V_\pi(s)$ jest zdefiniowana jako oczekiwana wartość skumulowanej nagrody otrzymanej, gdy zaczynając w stanie s, agent będzie działał zgodnie ze strategią π. Jest metryką określającą korzyść znalezienia się w danym stanie.

$$V_\pi(s) = \mathbb{E}_\pi \left[R \mid s_0 = s \right] = \mathbb{E}_\pi \left[\sum_{t=0}^{\infty} \gamma^t r_t \;\middle|\; s_0 = s \right] \tag{3.3}$$

gdzie r_t to wartość nagrody otrzymanej w kroku t, a $\gamma \in \langle 0, 1)$ (ang. *discount-rate*) to czynnik obniżający wartości nagród przyszłych gdyż są one mniej ważne niż nagrody natychmiastowe oraz aby zapewnić zbieżność ich sumy. W kontekście równania Bellmana formuła 3.3 jest równoznaczna z:

$$V_\pi(s) = \sum_{s' \in S} \mathrm{T}\big(s, \pi(s), s'\big) \Big(\mathrm{R}\big(s, \pi(s), s'\big) + \gamma V_\pi(s') \Big) \tag{3.4}$$

Podobnie definiuje się funkcję wartości akcji (ang. *action-value function*) $Q_\pi(s, a)$ jako oczekiwaną wartość skumulowanej nagrody otrzymanej, gdy zaczynając w stanie s i wybierając akcję a agent będzie dalej działał zgodnie ze strategią π. Zatem, analogicznie do funkcji V_π, funkcja Q_π określa korzyść z wyboru akcji a w stanie s.

$$Q_\pi(s, a) = \mathbb{E}_\pi \left[R \mid s_0 = s,\ a_0 = a \right] = \mathbb{E}_\pi \left[\sum_{t=0}^{\infty} \gamma^t r_t \;\middle|\; s_0 = s,\ a_0 = a \right] \tag{3.5}$$

Co jest równoważne kolejno z:

$$\begin{aligned} Q_\pi(s, a) &= \sum_{s' \in S} \mathrm{T}(s, a, s') \Big(\mathrm{R}(s, a, s') + \gamma Q_\pi\big(s', \pi(s')\big) \Big) \\ Q_\pi(s, a) &= \sum_{s' \in S} \mathrm{T}(s, a, s') \big(\mathrm{R}(s, a, s') + \gamma V_\pi(s') \big) \end{aligned} \tag{3.6}$$

3.1.4. Cel

Celem agenta jest znalezienie strategii optymalnej, której przestrzeganie skutkuje większą wartością skumulowanej nagrody niż dla każdej innej strategii. Wyprowadzenie strategii optymalnej jest możliwe dzięki funkcjom wartości, które wprowadzają relację częściowego porządku w zbiorze strategii [16]. W tym kontekście strategia π uznawana jest za lepszą lub równą strategii π' gdy:

$$\pi \geq \pi' \;\Leftrightarrow\; \forall_{s \in S} : V_\pi(s) \geq V_{\pi'}(s) \tag{3.7}$$

Optymalna strategia, oznaczana jako π^*, jest co najmniej równa wszystkim innymi strategiom:

$$\forall_\pi : V_{\pi^*}(s) \geq V_{\pi'}(s) \tag{3.8}$$

Strategia ta jest bezpośrednio kojarzona z optymalną funkcją wartości stanu, oznaczaną przez V^* i wyprowadzoną jako:

$$\forall_{s \in S} : V^*(s) = \max_\pi V_\pi(s) \tag{3.9}$$

Analogicznie posiada równoważną optymalną funkcję akcji, zwaną optymalną Q-funkcją, oznaczaną przez Q^* i definiowaną:

$$\forall_{s \in S,\, a \in A} : Q^*(s,a) = \max_\pi Q_\pi(s,a) \tag{3.10}$$

Optymalna Q-funkcja określa oczekiwaną wartość skumulowanej nagrody otrzymanej, gdy w stanie s po wyborze akcji a agent będzie działał zgodnie z optymalną strategią π^*.

3.1.5. Eksploracja i eksploatacja

Zrównoważenie eksploracji i eksploatacji środowiska przez agenta (ang. *exploration and exploitation trade-off*) to problem polegający na doskonaleniu się agenta w swojej strategii przy jednoczesnym dalszym odkrywaniu środowiska i próbowaniu nowych strategii. Wybór optymalnych akcji, według aktualnej wiedzy agenta, prowadzi do zdobywania największych nagród, jednak może blokować odkrycie strategii nowej, bliższej optymalnej. Z kolei wybór względnie nieoptymalnych akcji pozwala, kosztem mniejszej nagrody natychmiastowej, na szersze zbadanie środowiska. Może to doprowadzić do znalezienia strategii przynoszącej większe zyski w przyszłości. Dlatego uczenie ze wzmocnieniem wymaga przemyślanej metody eksploracji środowiska.

Proponowanym i szeroko stosowanym rozwiązaniem jest algorytm ε-*greedy*, gdzie parametr $\varepsilon \in (0, 1\rangle$ ustala proporcję między eksploracją a eksploatacją środowiska [17]. W każdym kroku treningu, z prawdopodobieństwem $1 - \varepsilon$, wybierana jest eksploatacja, czyli agent decyduje się na akcję, która, zgodnie z jego aktualną wiedzą skutkuje największą skumulowaną długo-terminową nagrodą. Równocześnie, z prawdopodobieństwem ε, wybierana jest eksploracja i agent wybiera akcję losowo. Zwykle wartość parametru ε jest progresywnie zmniejszana, dzięki czemu w początkowej fazie nauki agent głównie zbiera losowe doświadczenie, a w dalszej skupia się na doskonaleniu wybranej strategii.

3.2. Algorytmy

Jak wspomniane zostało w sekcji 3.1 algorytmy uczenia ze wzmocnieniem wykorzystują doświadczenie zebrane w czasie interakcji ze środowiskiem w celu odnalezienia optymalnej,

lub sub-optymalnej strategii. Osiągają to w różny sposób, zależnie od obiektu, który badają i wykorzystują do wyprowadzenia szukanej strategii. Wyodrębnione w ten sposób grupy zostaną opisane w tej sekcji.

3.2.1. *Policy-search*

Algorytmy badające bezpośrednio funkcję strategii (ang. *policy-search*) skupiają się na przeszukiwaniu jej przestrzeni (lub podprzestrzeni). Ponieważ problem staje się wówczas przykładem optymalizacji, zwykle wykorzystuje się techniki gradientowe. Strategia $\pi(a, s \mid \theta)$ jest parametryzowana przez zmienne θ, w przestrzeni których przeprowadza się *SGD*.

Techniki przybliżające funkcję strategii wykazują duże prawdopodobieństwo zbieżności do optimum, dzięki niewielkim i płynnym zmianom przebiegającym w przestrzeni prawdopodobieństw wyboru akcji [16]. Jednocześnie są podatne na eksplodujące gradienty (ang. *exploding gradient*), które mogą być zmniejszone poprzez implementację architektur agentów takich jak np. *Actor-Critic* [9].

Ponieważ strategia jest wypracowywana bezpośrednio w procesie nauki, to agenci z grupy *policy-search* są łatwi do późniejszego zastosowania. Wystarczy w danym kroku odczytać aktualny stan środowiska i korespondującą z nim akcję w strategii agenta.

3.2.2. *Value-based*

Algorytmy z tej grupy skupiają się na estymowaniu funkcji wartości stanu lub akcji dla pewnej strategii (algorytmy *on-policy*) lub optymalnej strategii (algorytmy *off-policy*). W zależności od modelu środowiska, jego wielkości oraz stopnia znajomości tworzących go elementów, algorytmy sekwencyjnie przybliżają się do zadanych wartości badanej funkcji. Jeśli model środowiska jest dokładnie znany, to możliwe jest iteracyjne przybliżanie funkcji wartości dla każdej pary stanu i akcji. Wymaga to obliczenia oczekiwanej wartości zwrotnej dla całej przestrzeni stanów i akcji. Jest to jednak niepraktyczne dla większych modeli decyzyjnych. W takich sytuacjach algorytmy uczenia ze wzmocnieniem przybliżają wartość oczekiwaną uśredniając jej wartość dla doświadczonych w procesie interakcji przykładów oraz wykorzystują metody interpolacji funkcji (np. sieci neuronowe), aby przybliżyć jej wartość dla niezbadanych podprzestrzeni stanu i akcji. Iteracyjne przybliżanie wartości funkcji przebiega zgodnie z metodami uczenia przez różnicę chwilową (ang. *temporal difference learning, TD-Learning*), zakładającymi dokonywanie drobnych korekt ogólnej wiedzy bazując na aktualnie doświadczonych wartościach. Jednym z najbardziej znanych algorytmów z tej grupy jest Q*-Learning*, który wraz ze swoimi rozszerzeniami został wykorzystany w tej pracy.

3.2.2.1. Q-*Learning*

Algorytm ten skupia się na estymacji optymalnej funkcji wartości akcji Q^*, z której bezpośrednio wyprowadzić można optymalna strategię π^*:

$$\pi^*(s) = \arg\max_a Q^*(s,a) \tag{3.11}$$

Gdzie Q^* wyprowadzono z pierwszej formy funkcji Q przedstawionej w równaniu 3.6:

$$Q^*(s,a) = \sum_{s' \in S} \mathrm{T}(s,a,s') \left(\mathrm{R}(s,a,s') + \gamma \max_{a'} Q^*(s',a') \right) \tag{3.12}$$

W algorytmie Q-*Learning* agent reprezentuje funkcję Q w formie tabelarycznej. Inicjuje jej wartości początkowe a następnie w każdym kroku t interakcji ze środowiskiem obserwuje stan s_t środowiska, wybiera akcję $a_t = \pi(s_t)$ zgodnie z dowolną strategią π, otrzymuje nagrodę r_t i obserwuje kolejny stan środowiska s_{t+1}. Następnie przeprowadza aktualizację wartości w tabeli zgodnie z równaniem Bellmana:

$$Q_\pi(s_t,a_t) \leftarrow Q_\pi(s_t,a_t) + \alpha \cdot \underbrace{\left(r_t + \gamma \max_a Q_\pi(s_{t+1},a) - Q_\pi(s_t,a_t) \right)}_{\text{różnica chwilowa (}TD\text{)}}, \tag{3.13}$$

gdzie $\alpha \in (0,1\rangle$ to stała ucząca (ang. *learning rate, LR*) odpowiedzialna za prędkość uczenia. Zakładając strategię π zapewniającą ciągłe odwiedzanie i aktualizowanie wszystkich par stanu i akcji obliczana reprezentacja funkcji Q wykazuje zbieżność do optymalnej Q^* z prawdopodobieństwem równym 1 [16]. Przykładem takiej strategii jest zależna od aktualnie przechowywanej wartości Q z niezerową szansą eksploracji, która została opisana w algorytmie ε-*greedy* w sekcji 3.1.5.

3.2.3. *Model-driven*

Ostatnia grupa algorytmów uczy się modelu środowiska, czyli estymuje opisującą go dynamikę. Zbierane w trakcie interakcji doświadczenie pozwala na przybliżanie funkcji przejścia T oraz nagrody R. Wiedza o modelu środowiska może być dalej użyta w części z opisanych wcześniej algorytmów.

3.3. Zaawansowane struktury i mechanizmy

W tej pracy wykorzystano opisany wcześniej algorytm Q-*Learning* ze względu na udowodnioną zbieżność i wiele przykładów skutecznego zastosowania [12]. Dodatkowo badane były bardziej zaawansowane warianty tegoż algorytmu oraz struktury pozwalające na przyspieszenie i poprawę procesu nauki.

3.3.1. *Deep Q-Learning*

W podstawowej wersji funkcja Q jest modelowana w formie tabelarycznej. W przypadku dużej przestrzeni stanów i akcji możliwe jest, że niektóre pary zostaną zaktualizowane dopiero po bardzo długiej nauce. W *Deep Q-Learning* rozwiązuje się to poprzez wykorzystywanie sieci neuronowych do modelowania funkcji Q. Pozwalają one aproksymować wartości dla całej przestrzeni bazując jedynie na zdobytym doświadczeniu.

Jednak w związku z dużą wariancją kolejnych obserwacji metoda ta wykazuje niestabilność procesu nauki i traci pewność zbieżności. Drobne zmiany dokonywane sekwencyjnie na nieliniowej reprezentacji Q (a nie tylko na jednej wartości w tabeli) mają wpływ na całą aktualną strategię agenta i wprowadzają wspomnianą korelację między wartością docelową a aktualną. W celu przywrócenia stabilności nauki wykorzystuje się m.in. opisane dalej metody *Experience Replay* oraz *Double Deep Q-Learning*.

3.3.2. *Experience Replay*

Metoda ta jest zainspirowana mechanizmem biologicznym polegającym na korzystaniu z przeszłych obserwacji na równi z ostatnimi w procesie podejmowania decyzji. Zakłada przechowywanie ostatnich krotek doświadczeń $(s_t, a_t, r_t, s_{t+1})_{t\geq 0}$ w strukturze pamięci zwanej buforem (ang. *Replay Buffer*). Podczas treningu, w równych interwałach wybiera się z bufora losową próbkę, którą wykorzystuje się do aktualizacji wartości funkcji Q. Takie rozwiązanie pozwala pozbyć się wspomnianej wariancji oraz zwiększa stabilność nauki dzięki wielokrotnemu wykorzystaniu zebranego doświadczenia w treningu.

Dalszym rozszerzeniem tej metody jest wykorzystanie bufora priorytetowego (ang. *Prioritized Replay Buffer*), który dodatkowo dla każdej krotki doświadczenia oblicza jej wagę interpretowaną jako względną przydatność w procesie nauki i odpowiedzialną za prawdopodobieństwo wylosowania jej z bufora. Wartość ta jest zależna od różnicy chwilowej z nią skojarzonej. Metoda ta wykazuje jeszcze szybszą zbieżność nauki [6].

W pewnych środowiskach występują grupy stanów częściej odwiedzanych. Wykorzystując podstawową wersję bufora pamięci będą one użyte w procesie nauki więcej razy skutkując dla nich dobrą estymacją wartości Q w przeciwieństwie do niedokładnych przybliżeń dla rzadko odwiedzonych stanów. Propozycją adresującą opisany problem jest struktura nazwana pałacem pamięci (ang. *Memory Palace*), w której krotki doświadczenia dzielone są na rozłączne zbiory-pokoje względem ustalonej zasady [19]. Losując próbkę doświadczeń wybierana jest jednakowa liczba krotek z każdego z pokoi. Uważny wybór reguły podziału pozwoli dokładniej przybliżyć wartości Q dla każdej podgrupy stanów.

3.3.3. *Double Q-Learning*

W równaniu 3.13 największa możliwa wartość skumulowanej nagrody jest estymowana przy pomocy tej samej funkcji Q, która jest odpowiedzialna za aktualną strategię agenta. W niestabilnych środowiskach może to powodować przeszacowaniem tej wartości, co dalej spowalnia proces uczenia. Adresując ten problem zaproponowany został wariant zwany *Double Q-Learning* [4], w którym inna strategia wykorzystywana jest do estymacji wartości nagrody niż do wyborów agenta.

Metoda ta została również wykorzystana w *Deep Q-Learning* skutkując powstaniem wariantu *Double Deep Q-Learning* charakteryzującego się stabilniejszym i pewniejszym procesem uczenia [5]. Wykorzystuje się w niej dwie sieci neuronowe aproksymujące Q. Pierwsza z nich, odpowiadająca za strategię agenta, jest aktualizowana okresowo na losowej próbce krotek doświadczeń pobranych z bufora pamięci. Druga, służąca jako estymacja przyszłej nagrody, jest nadpisywana wartościami pierwszej z ustaloną częstotliwością. Częstotliwość kopiowania jest o kilka rzędów większa niż częstotliwość aktualizacji.

4. Analiza teoretyczna problemu

W tym rozdziale przedstawiony został problem przewożenia pasażerów w systemach sterowania windą jako problem decyzyjny badany i optymalizowany w niniejszej pracy. Opisana została jego złożoność, oraz wspomniany został jego model teoretyczny wraz ze skojarzonymi z nim istniejącymi rozwiązaniami.

4.1. Własności systemu kontroli windy

System kontroli windy składa się z jednej lub więcej wind operujących w wielopiętrowym budynku. W drugim przypadku określany jest jako *group control* [15]. System rejestruje wezwania windy z pięter oraz wezwania z kabin i obsługuje je zgodnie z wybraną strategią. Wezwania z pięter są generowane przez fizyczne przyciski znajdujące się na piętrach, niekiedy z rozróżnieniem na kojarzone z zamiarem jazdy w górę oraz z chęcią jazdy w dół. Wewnątrz kabiny windy znajdują się przyciski zatrzymania wyzwalające wezwania jazdy na konkretne piętro. Celem systemu kontroli windy jest obsługa jej pasażerów w sposób optymalizujący:

– średni oraz maksymalny czas oczekiwania pasażerów na dojazd do celu,

– przebyty dystans przez każdą z wind.

Problem znalezienia strategii w tym kontekście optymalnej należy do szerszej grupy zagadnień transportu i zapotrzebowania w czasie rzeczywistym (ang. *online-dial-a-ride*).

4.2. Klasyczne strategie kontroli

Zadanie stawiane przed sterownikiem systemu windy może być zdefiniowane następująco: dla N pasażerów, chcących dostać się z piętra startowego s na piętro docelowe d: $(s_1, d_1), (s_2, d_2) \ldots (s_N, d_N)$, należy wyznaczyć ciąg odwiedzonych pięter $p_1, p_2 \ldots p_k$ w taki sposób, aby dla każdego pasażera jego piętro startowe s_i poprzedzało jego piętro docelowe d_i. Jeśli dodatkowo należy znaleźć ciąg o minimalnej długości, to problem staje się $\mathcal{NP}$-zupełny przez redukcję z problemu cyklicznego pokrycia wierzchołkowego, co zostało udowodnione

przez B.Seckinger i J.Koehler w 1999 roku [14]. Podobnie problem adresowany w tej pracy może być zbliżony do wariantów problemów Komiwojażera lub Marszrutyzacji, jednak żaden z nich nie modeluje go w pełni. Wynika to z faktu zależności kosztu przejazdu nie tylko od kolejności obsłużenia pasażerów, ale również sposobu ich grupowania. Dodatkowo wspomniane problemy operują na środowiskach niezmiennych i zakładają pełną wiedzę o nich, a w systemie sterowania windą pasażerowie pojawiają się dynamicznie i pełny stan systemu jest nieznany. Z tego powodu powstały algorytmy heurystyczne obsługi systemu windy w czasie rzeczywistym. Poniżej zostały opisane te z nich, które mogą być wykorzystane w systemie z jedną windą.

Algorytm najdłuższej kolejki

Algorytm najdłuższej kolejki (ang. *Longest Queue First, LQF*) wykorzystuje kolejkę celem ustalenia, które wezwanie obsłużyć w danej chwili. Kiedy pasażer wzywa windę z piętra lub z kabiny, wezwanie jest dodawane na koniec kolejki. Wezwania są obsługiwane od najdłużej oczekującego. Na piętrach znajduje się tylko jeden przycisk, czyli pasażer nie może sprecyzować czy jedzie na piętro wyższe czy niższe. Dodatkowo wezwania powstałe skutkiem wciśnięcia przycisku wewnątrz kabiny mają większy priorytet niż te wygenerowane z piętra. Dzięki temu pasażerowie znajdujący się aktualnie w windzie zostaną obsłużeni zanim winda zajmie się zabraniem nowego pasażera [2].

Algorytm kolektywny i ostatniej misji

Algorytm kolektywny (ang. *Collective Algorithm, CA*) jest rozszerzeniem algorytmu najdłuższej kolejki. Zakłada w trakcie obsługi najdłużej oczekującego wezwania zatrzymywanie się na piętrach pośrednich, dla których aktywne jest wezwanie z kabiny lub z piętra. Jeżeli dodatkowo zatrzymuje się na piętrze jedynie, gdy kierunek wezwania z tego piętra jest zgodny z aktualnym kierunkiem poruszania się windy, to nazywany jest algorytmem ostatniej misji (ang. *Last Mission, LM*). W tym wariancie na piętrach znajdują się po dwa przyciski wezwania pozwalające pasażerom sprecyzować kierunek podróży.

4.3. Nowoczesne rozwiązania

Rozwiązanie zaproponowane przez Imasaki i in. wykorzystuje rozmytą sieć neuronową w predykcji rozkładu czasu oczekiwania pasażerów dla zróżnicowanych parametrów kontrolnych [8]. Przestrzeń stanów wejściowych jest podzielona na podprzestrzenie za pomocą zasad logiki rozmytej. Kontroler wspomaga swoje decyzje predykcjami sieci, jednak nie jest sprecyzowane jaki algorytm kontroli jest najwyżej.

Markon wraz z współpracownikami opracowali system trenujący sieć neuronową do przeprowadzania natychmiastowej alokacji wind [11]. Przeprowadzili trening sieci w trzech fazach. Pierwsza wykorzystywała uczenie nadzorowane z danymi zebranymi z systemu sterowanego przez istniejące rozwiązania firmy *Fujitec*. Wybrane wagi sieci zostały zamrożone jako odpowiedzialne za reprezentację zasad działania systemu kontroli. W drugiej fazie reszta sieci była douczana, aby naśladować kontroler. W trzeciej fazie znów cała sieć była douczana w emulatorze systemu w sposób zwiększający jej osiągi dla wygenerowanej próbki ruchu pasażerów, co może być rozumiane jako uproszczona forma uczenia ze wzmocnieniem. Warstwa wejściowa sieci składała się z 25 neuronów dla każdej windy, a wyjściowa z jednego dla każdej windy. Wezwania z pięter były przydzielane do windy z największą wartością aktywacji odpowiadającego neuronu wyjściowego. System został przetestowany w emulatorze budynku z 15 piętrami i 6 windami, na którym uzyskał drobną poprawę w odniesieniu do istniejącego kontrolera.

W poźniejszych latach również uczenie ze wzmocnieniem doczekało się swoich implementacji. Crites i Barto wykorzystali koncepcję *Deep RL* w systemie sterowania jedną windą w 10-piętrowym budynku w czasie szczytu ruchu w dół [1]. Pasażerowie obierali trasy z piętrem najniższym jako docelowym oraz pojawiali się zgodnie z rozkładem zależnym od fazy symulacji. Trening trwał 4 dni i w tym czasie algorytmy przebywały w symulowanym systemie ponad 60 tysięcy godzin. Wytrenowane algorytmy odnotowały widocznie lepsze wyniki od znanych algorytmów heurystycznych.

5. Projekt i metodyka rozwiązania

W tym rozdziale przedstawiona została metodyka wybrana w trakcie pracy. Opisane zostały kolejne kroki i przybliżona argumentacja stojąca za wyborem ich realizacji.

5.1. Symulacja

Decyzja o zastosowaniu w pracy algorytmów uczenia ze wzmacnianiem wymusiła potrzebę stworzenia środowiska modelującego system sterowania windą. Zdecydowano się na zaimplementowanie uniwersalnego symulatora windy. Takie rozwiązanie pozwoliło w dalszych etapach pracy na enkapsulowanie go w szerokim spektrum środowisk *RL* różniących się skomplikowaniem, reprezentacją stanu i metodą nagradzania.

Symulator parametryzowany jest przez następujące zmienne:

– liczba pięter, $N_p \in \mathbb{Z}_+$

– pojemność windy, $P_w \in \mathbb{Z}_+$

– pojemność piętra, $P_p \in \mathbb{Z}_+$

Symulator jest systemem czasu dyskretnego, co oznacza, że możliwa interakcja następuje w kwantach czasu zwanych krokami symulacji. Jest to uproszczenie w stosunku do systemu rzeczywistego przy dalszym go odwzorowywaniu. Pasażerowie mogą być wprowadzeni do systemu w aktualnym kroku symulacji. Pasażer dodawany jest na wybranym piętrze na koniec jej kolejki (jeśli nie jest pełna) i z określonym piętrem docelowym, różnym od startowego. W każdym kroku symulacji dostępne są następujące akcje:

A. ruch windy o jedno piętro w górę (niemożliwe na najwyższym piętrze);

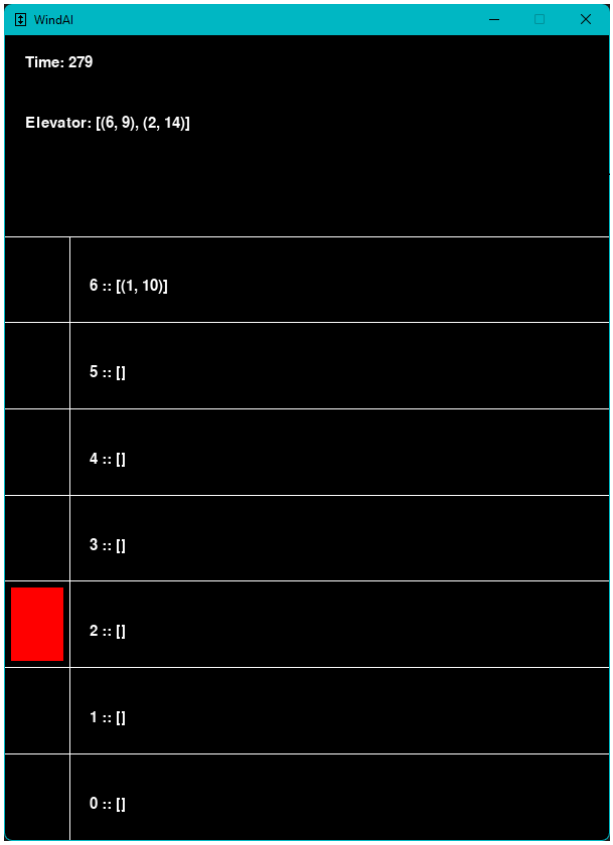
B. ruch windy o jedno piętro w dół (niemożliwe na najniższym piętrze);

C. zatrzymanie windy, które łączy się z obsługą pasażerów i przebiega w dwóch krokach:

   1. wszyscy pasażerowie przebywający w windzie, których piętrem docelowym jest aktualne piętro, wysiadają i kończą tym samym swoją podróż,

2. na aktualnym piętrze wsiada z kolejki do windy maksymalnie tylu z pierwszych pasażerów ile jest miejsca w windzie.

Symulator został zaimplementowany w języku *Python* z wykorzystaniem struktur danych wbudowanych oraz oferowanych przez bibliotekę *NumPy*, które pozwoliły na zapewnienie optymalności obliczeń. Prosta wizualizacja symulowanego systemu windy została stworzona przy wykorzystaniu biblioteki *pygame*. Przykładowy jej widok przedstawiono na rysunku 5.1.



Rys. 5.1. Wizualizacja symulacji dla systemu z 7 piętrami. Aktualne piętro windy ukazuje położenie czerwonego prostokątu w siatce pięter. Pasażerowie będący w windzie widoczni są w tablicy Elevator. Podobnie na każdym piętrze wyświetlany jest stan kolejki pasażerów. Pasażerowie w wizualizacji reprezentowani są przez krotkę (D, T), gdzie D jest piętrem docelowym pasażera, a T – czasem jego przebywania w systemie. W widocznym 279 kroku symulacji w windzie znajdują się pasażerowie jadący na 6 i 2 piętro, odpowiednio będący w systemie 9 i 14 kroków. Poza tym na piętrze 6 od 10 kroków czeka pasażer chcący dostać się na piętro 1.

Źródło: opracowanie własne

5.2. Rozkład pasażerów

W celu stworzenia modelu środowiska adekwatnie obrazującego rzeczywisty problem sterowania windą nie wystarczy tylko symulacja. Potrzebny jest również rozkład pojawiania się pasażerów w zależności od czasu i wybranej trasy. Do przybliżenia takiego rozkładu wykorzystano dane pochodzące z odczytów systemu kontroli windy w inteligentnym budynku firmy ASTOR sp. z o.o.. Wykorzystano w nim zaawansowane narzędzie *Historian*, opracowane przez AVEVA i służące do zbierania i przechowywania danych przemysłowych. *Historian* pozwala na optymalne odczytywanie i zapisywanie danych napływających z dużą częstotliwością, wykorzystując nowatorską metodę reprezentacji danych w formie zdarzeń zmian wartości. Dzięki dojrzałemu podejściu do danych i znając ich wartość rosnącą wraz z czasem, firma ASTOR już ponad 10 lat temu rozpoczęła archiwizowanie odczytów z czujników w całym budynku (ponad 300 tysięcy parametrów), w tym również pobieranych z zainstalowanego systemu kontroli windy (51 parametrów).

W rozważanym budynku jest 7 pięter (od piętra -1 do 5) oraz jedna winda. Wewnątrz windy dostępne są przyciski zatrzymania dla każdego z 7 pięter. Na każdym piętrze znajdują się po dwa przyciski wezwań, jeden kojarzony z zamiarem jazdy na piętro wyższe, drugi na piętro niższe (nie odnosi się to do pięter -1 oraz 5 gdzie jest po jednym przycisku). Dostępne parametry można podzielić na opisowe i skumulowane. Te pierwsze dotyczą wartości konkretnej zmiennej (a raczej, ze względu na charakter bazy danych, jej nowej, zmienionej wartości). Drugie są zliczoną dla danego okresu czasu (dzień lub przedział trwający od początku zbierania danych) liczbą zarejestrowanych wartości parametru o zadanej wartości. Drugie parametry nie opisują dynamiki systemu, nie będą więc poruszone w tej pracy. Rozważanymi parametrami w dalszej analizie będą:

– aktualne położenie windy, wartość ze zbioru $\{-1, 0, 1, 2, 3, 4, 5\}$, zmiana piętra jest rejestrowana podczas przejazdu;

– piętro postoju windy, wartość ze zbioru $\{-1, 0, 1, 2, 3, 4, 5\}$, jednak w przeciwieństwie do piętra położenia windy zmiana piętra postoju jest rejestrowana dopiero w momencie zatrzymania windy,

– aktualny stan przycisków wewnątrz windy (7 przycisków) oraz na piętrach (12 przycisków), gdzie: 0 – nieaktywny, 1 – aktywny.

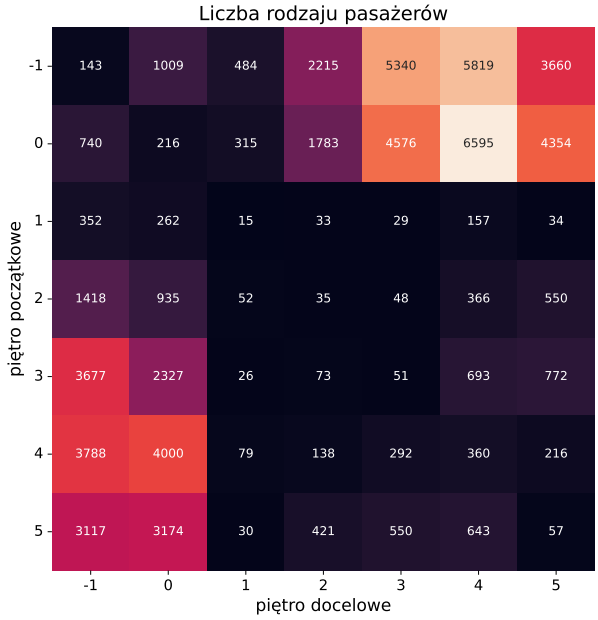
Dla wybranych parametrów, przy wykorzystaniu biblioteki *pymmsql*, pobrano z lokalnej bazy *Historian* w firmie ASTOR dane zmian ich wartości na przestrzeni od sierpnia 2018 do sierpnia 2019. Zostały one zapisane do struktury danych *DataFrame* z biblioteki *pandas*. Wynikowy *DataFrame* posiada kolumny:

– *Timestamp* – data zarejestrowanej zmiany wartości parametru,

– *TagName* – nazwa parametru,

– *Value* – wartość po zmianie.

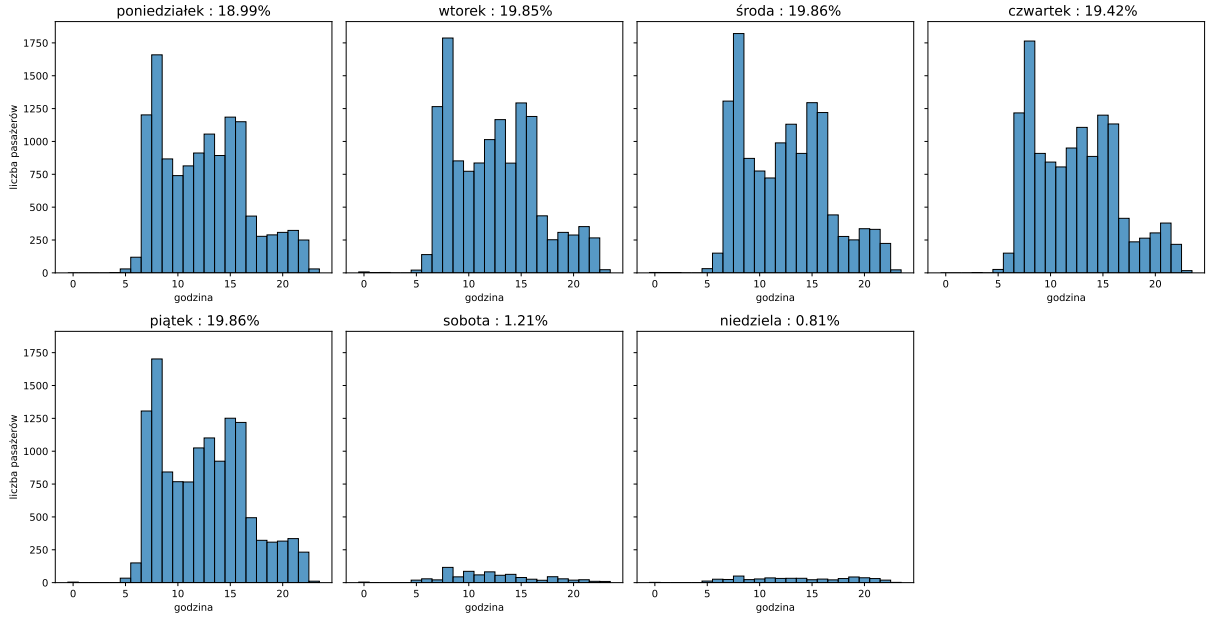
W kolejnym kroku zliczono i sklasyfikowano pasażerów korzystających z windy w badanym okresie. Założono w tym celu, że użytkownik wchodząc do windy stojącej na piętrze A natychmiast wybiera i wciska przycisk zatrzymania Z_B wewnątrz kabiny odpowiedni dla jego docelowego piętra B. Takie założenie pozwoliło, przechodząc sekwencyjnie po rejestrowanych wartościach zmian parametrów, każdą zmianę aktywności przycisku Z_x zatrzymania wewnątrz windy skojarzyć z konkretnym pasażerem jadącym na piętro x z piętra y, na którym ostatnio zarejestrowano zatrzymanie windy.



Rys. 5.2. Liczba wykrytych pasażerów ze względu na obraną przez nich trasę. Na osi pionowej piętro początkowe, na osi poziomej piętro docelowe. Najwięcej pasażerów decydujących się na jazdę windą zaczyna lub kończy podróż na piętrach niższych. Najczęściej obieraną trasą jest z parteru na piętro 4, gdzie pracuje najwięcej osób. Więcej pasażerów windą wyjeżdża niż zjeżdża (39 tysięcy do 26 tysięcy). Niezerowe wartości na przekątnej wynikają z obranych założeń oraz błędnych wyborów użytkowników windy.

Źródło: opracowanie własne

Tym sposobem zliczono 65,142 pasażerów sklasyfikowanych ze względu na trasę (para pięter startowe i docelowe) oraz dnień tygodnia i godzinę podróży. Wartości zostały zapisane do 4-wymiarowej macierzy biblioteki *numpy* o wymiarach (7 dni tygodnia) x (24 godziny) x (7 pięter



Rys. 5.3. Rozkład liczby pasażerów w ciągu dnia w kolejne dni tygodnia. W tytule każdego z 7 wykresów obok nazwy dnia tygodnia widnieje procentowy udział liczby pasażerów w tym dniu w stosunku do wszystkich wykrytych. W weekend jest znacząco mniejszy ruch niż w dniach roboczych.

Źródło: opracowanie własne

startowych) x (7 pięter docelowych). Następnie sprawdzono, że średni czas trwania przejazdu windy między dwoma piętrami to około 4 sekundy. Obliczono średnią częstotliwość pojawiania się pasażerów na jeden taki interwał według obranej trasy, dnia tygodnia i godziny dzieląc liczbę pasażerów w macierzy przez liczbę interwałów przypadających na daną godzinę w danym dniu tygodnia. Powtórzono opisany proces również dla zagregowanych macierzy względem wymiarów dnia i godziny lub tylko dnia. Otrzymano odpowiednio: średnią częstotliwość pojawiania się pasażerów ze względu na obraną trasę oraz ze względu na trasę i godzinę. Dzięki temu stworzono rozkłady prawdopodobieństwa pojawiania się pasażerów w dyskretnej wersji czasowej badanego systemu dla różnych poziomów ogólności, które wykorzystano później w reprezentacjach środowiska.

5.3. Model środowiska

W niniejszej pracy utworzono wiele różnorodnych środowisk uczenia ze wzmocnieniem. Każde z nich imitowało system sterowania windą w sposób umożliwiający agentowi interakcję, zgodnie z przyjętą strukturą opisaną w podsekcji 3.1.1. Za modelowanie dynamiki odpowiedzialny był zawarty w środowisku symulator, który wraz z wybranym rozkładem pojawiania

się pasażerów definiował funkcję przejścia T oraz zbiór dostępnych akcji A. W każdym środowisku konstruowana była również funkcja nagrody R oraz mapowanie stanu symulatora na stan środowiska formujące przestrzeń stanów S.

W zaproponowanej architekturze środowisko jest jednoznacznie definiowane przez parametry symulatora, wybrany rozkład pojawiania się pasażerów, funkcję R oraz sposób reprezentacji stanu. W dalszej części opisane zostaną instancje tych cech dla systemu o N piętrach.

5.3.1. Reprezentacja stanu

Stan środowiska odczytywany przez agenta w kroku t jest reprezentowany przez wektor stanu środowiska x_t i ustalany na podstawie aktualnego stanu symulatora. Wykorzystane w tej pracy konstrukcje tego wektora dla przyjętego wyżej systemu zostały wyszczególnione poniżej:

1. Pozycja windy, stan przycisków

W podstawowej metodzie reprezentacji stanu jego wektor w kroku t ma postać:

$$x_t = [p_t,\ Z_t^1, Z_t^2\ \ldots\ Z_t^N,\ W_t^1, W_t^2\ \ldots\ W_t^N]^T,$$

gdzie:

- $p_t\ \in \{1, 2\ ...\ N\}$ to pozycja windy w kroku t;
- $Z_t^i\ \in \{0, 1\},\ i = 1, 2 \ldots N$ to stan przycisku zatrzymania windy na piętrze i w kroku t, gdzie wartość 1 oznacza, że w windzie znajduje się co najmniej jeden pasażer jadący na dane piętro, a 0 w przeciwnym przypadku;
- $W_t^i\ \in \{0, 1\},\ i = 1, 2 \ldots N$ to stan przycisku wezwania windy na piętrze i w kroku t, gdzie wartość 1 oznacza, że na danym piętrze czeka co najmniej jeden pasażer, a 0 w przeciwnym przypadku.

2. Pozycja windy, stan przycisków góra/dół

W tej reprezentacji wektor stanu z punktu (1) jest rozszerzony o zróżnicowanie wezwań na piętrze na kategorie: jazda w górę, jazda w dół. W rezultacie:

$$x_t = [p,\ Z_t^1, Z_t^2\ \ldots\ Z_t^N,\ D_t^2, D_t^3\ \ldots\ D_t^N,\ G_t^1, G_t^2\ \ldots\ G_t^{(N-1)}]^T,$$

gdzie:

- $D_t^i\ \in\ \{0, 1\},\ \ i\ =\ 2 \ldots N$ to stan przycisku wezwania windy w dół na piętrze i, w kroku t;
- $G_t^i\ \in \{0, 1\},\ i = 1 \ldots (N-1)$ to stan przycisku wezwania windy w górę na piętrze i, w kroku t.

3. Pozycja windy, godzina w symulacji, stan przycisków góra/dół

W tej reprezentacji dodawany jest element informujący o aktualnej godzinie w symulowanym systemie. Był wykorzystywany w środowiskach, dla których rozkład pasażerów był zróżnicowany względem godziny.

$$x_t = [\, p_t,\ Z_t^1\ \ldots\ Z_t^N,\ T(D_t^2)\ \ldots\ T(D_t^N),\ T(G_t^1)\ \ldots\ T(G_t^{(N-1)})\,]^T,$$

gdzie $h_t \in \{0, 1\ \ldots\ 23\}$ jest aktualną godziną mającą wpływ na prawdopodobieństwo pojawiania się pasażerów w systemie.

4. Pozycja windy, godzina w symulacji, czas wciśnięcia przycisków góra/dół

W tej reprezentacji zamiast stanu przycisku widnieje długość jego aktywności, rozumiana jako liczba kroków symulacji, od których jest on nieprzerwanie aktywny, a 0 jeśli jest nieaktywny. Ma to wpływ na znaczące zwiększenie konstruowanego wówczas zbioru S stanów środowiska.

$$x_t = [\, p_t, h_t,\ T(Z_t^1)\ \ldots\ T(Z_t^N),\ T(D_t^2)\ \ldots\ T(D_t^N),\ T(G_t^1)\ \ldots\ T(G_t^{(N-1)})\,]^T,$$

gdzie T jest funkcją postaci:

$$T(P_t) = \begin{cases} 1 + T(P_{t-1}) & \text{jeśli } P_t = 1, \\ 0 & \text{jeśli } P_t = 0 \vee t = 0, \end{cases}$$

dla przycisku P w kroku t.

5.3.2. Funkcja nagrody

Konstrukcja funkcji nagrody jest kluczowa w procesie tworzenia środowiska. Definiuje ona bowiem optymalną strategię, która maksymalizuje jej skumulowaną wartość w czasie. Jest jedyną dostępną agentowi metryką jakości akcji w danym stanie w kontekście nieznanego i abstrakcyjnego celu odgórnie zadanego w środowisku. Nieostrożne jej wybranie może skutkować dalekimi od planowanych zachowaniami agenta.

Nagroda jest obliczana w następstwie wybranej przez agenta akcji i zależna od niej oraz od wynikowego stanu symulatora. Jej wartość może być również ujemna i jest wtedy zwykle interpretowana jako kara. Badane w tej pracy funkcje nagrody R są kombinacjami liniowymi elementów, z których każdy ma swoją interpretację opisu jakości akcji bądź stanu:

A. Kara za liczbę wciśniętych przycisków

$$A(t) = -\sum^{P} P_t,$$

gdzie P to zbiór rozważanych przycisków. Zbiór ten jest, wewnątrz środowiska, zgodny z tym zawartym w metodzie reprezentacji stanu. Maksymalizacja wartości tego elementu jest równoznaczna ze zmniejszaniem liczby oczekujących pasażerów.

B. Kara za czas wciśniętych przycisków

$$B(t) = -\sum^{P} T(P_t)$$

Maksymalizacja wartości sumy skutkuje zmniejszaniem liczby oczekujących pasażerów, a zwłaszcza tych najdłużej przebywających w systemie.

C. Kara za ruch windy

$$C(t) = \begin{cases} 0 & \text{jeśli } a_t \text{ to zatrzymanie windy,} \\ -1 & \text{jeśli } a_t \text{ to jazda w górę lub w dół} \end{cases}$$

Użycie tego elementu miało zapewnić zmniejszenie niepotrzebnych ruchów windy.

D. Kara za nielegalny ruch

$$D(t) = \begin{cases} -1 & \text{jeśli } a_t \text{ jest nielegalną akcją,} \\ 0 & \text{jeśli } a_t \text{ jest legalną akcją} \end{cases}$$

Nielegalny ruch w kontekście symulatora to wybranie akcji niemożliwej w modelowanym systemie, czyli jazdy w górę na piętrze najwyższym oraz jazdy w dół na najniższym. W takiej sytuacji symulator nie zmienia położenia windy ani nie wypuszcza pasażerów, jedynie inkrementuje liczbę kroków (jest więc równoznaczny zatrzymaniu się bez otwarcia drzwi).

Karanie agenta za akcję nielegalną jest jednym z rozwiązań zagadnienia zależności dostępnych akcji od stanu środowiska. Takie rozwiązanie, zwłaszcza gdy ograniczeń jest niewiele, upraszcza implementację.

E. Nagroda za wypuszczenie pasażera

$$E(t) = p_t,$$

gdzie p_t to liczba wypuszczonych pasażerów w kroku t. Zwiększanie wartości tego elementu kojarzone jest z obsługiwaniem jak największej liczby pasażerów.

F. Nagroda/kara za zatrzymanie środowiska

$$F(t) = \begin{cases} 1 & \text{jeśli } x_t \in \Omega\ , \\ 0 & \text{w przeciwnym wypadku} \end{cases}$$

Zatrzymanie środowiska następuje, gdy znajdzie się w dowolnym stanie ze zbioru stanów $\Omega \subset S$, który jest konstruowany dla danego środowiska. Stany te są rozumiane jako docelowe lub, wręcz odwrotnie, niedopuszczalne. Dla przykładu stanem docelowym może być obsłużenie wszystkich pasażerów a stanem niedopuszczalnym, gdy pewien pasażer czeka co najmniej τ kroków.

G. Nagroda za zmianę wartości sumy czasu wciśnięcia przycisków

$$G(t) = \sum^{P} \big(T(P_{t-1}) - T(P_t)\big) = B(t) - B(t-1)$$

Motywacją przy tworzeniu tego elementu funkcji R była chęć nagradzania agenta za każdą pozytywną zmianę w skumulowanej wartości czasu oczekiwania pasażerów. Intuicyjnie było to skojarzone z jak najczęstszym wypuszczaniem pasażerów. Jest to jednak przykład nieostrożnie wybranego parametru, którego efekty, odwrotne do zamierzonych, zostaną opisane w dalszej części pracy.

Ostateczna forma

Podsumowując funkcja nagrody ma postać:

$$\mathrm{R}(t) = s_a \cdot A(t) + s_b \cdot B(t) + s_c \cdot C(t) + s_d \cdot D(t) + s_f \cdot F(t) + s_g \cdot G(t),$$

gdzie współczynniki $s_a, s_b \ldots s_g$ są liczbami rzeczywistymi definiowanymi dla każdego środowiska. Przybierały wartości nieujemne, z wyjątkiem s_f, którego znak był zależny od konstrukcji i interpretacji zbioru Ω.

Dodatkowo, wartość współczynnika s_d była dostatecznie duża, tak aby wybranie nielegalnej akcji zawsze skutkowało niższą nagrodą natychmiastową niż wynikającą z dowolnej akcji legalnej.

5.3.3. Rozkład pojawiania się pasażerów

Pasażerowie są wprowadzani do symulacji w każdym jej kroku zgodnie z ustalonym rozkładem prawdopodobieństwa określonym w przestrzeni par pięter (początkowego i docelowego, bez powtórzeń). Wraz z symulowaną dynamiką systemu rozkład ten modeluje funkcję T przejścia środowiska.

Prawdopodobieństwo pojawienia się pasażera jadącego z piętra początkowego p_1 na piętro docelowe p_2 określone jest przez $Pr(p = p_1, d = p_2)$. W tej pracy wykorzystane zostały niżej wymienione rozkłady.

1. Pojedynczy pasażer

Najprostszy rozkład pojawiania się pasażerów zakładał dodanie w kroku 0 symulacji pojedynczego pasażera jadącego z piętra najwyższego na najniższe. W kolejnych krokach prawdopodobieństwo pojawienia się pasażera dla dowolnej pary pięter było zerowe:

$$Pr(p, d) = 0$$

2. Ruch w dół

Rozkład ten modeluje uproszczony ruch pasażerów w godzinach wieczornych, kiedy pracownicy kończąc pracę zjeżdżają windą ze swoich pięter na sam dół. Prawdopodobieństwo określone jest jako:

$$Pr(p, d) = \begin{cases} 0 & \text{jeśli } d \neq 0, \\ \frac{f}{N-1} & \text{jeśli } d = 0 \end{cases}$$

W ten sposób prawdopodobieństwo pojawienia się dowolnego pasażera w pojedynczym kroku, równoznaczne z częstotliwością pojawiania się pasażerów, to ustalona wartość f.

3. Równomierny ruch

Pasażerowie pojawiają się z oczekiwaną częstotliwością f równomiernie rozłożeni względem obranej trasy:

$$Pr(p, d) = \frac{f}{N(N-1)}$$

4. Rozkład z budynku ASTOR według trasy

Prawdopodobieństwo pojawienia się pasażera dla danej pary pięter jest określone zgodnie z obliczoną macierzą $\mathbf{T}$ średnich częstotliwości pojawiania się pasażerów obierających określoną trasę w budynku ASTOR, która została opisana w sekcji 5.2.

Średni okres pojawienia się dowolnego pasażera w badanym systemie to około $5\frac{1}{3}$ minuty, które odpowiadają 80 krokom symulacji (80 interwałów 4 sekundowych). W celu uzyskania większej dynamiki systemu, przy jednoczesnym podtrzymaniu charakteru rozkładu, w pracy wykorzystano wersje macierzy częstotliwości przemnożonych 8 lub 16-krotnie. Dzięki temu uzyskano średnie okresy odpowiednio 10 i 5 kroków:

$$Pr(p, d) = 8 \cdot \mathbf{T}_{p,d}$$
$$Pr(p, d) = 16 \cdot \mathbf{T}_{p,d}$$

5. Rozkład z budynku ASTOR według godziny i trasy

W tym rozkładzie, poza obieraną trasą, prawdopodobieństwo pojawienia się pasażera ustala aktualna godzina w symulacji h_t. Wartość prawdopodobieństwa jest zgodna z obliczoną macierzą $\mathbf{H}$ średnich częstotliwości pojawiania się pasażerów w danej godzinie obierających określoną trasę w budynku ASTOR opisaną w sekcji 5.2.

$$Pr(h_t, p, d) = \mathbf{H}_{h_t,p,d}$$

Tym razem nie mnożono wartości z obliczonej macierzy. W rezultacie otrzymano średni okres równy około 80 kroków, który pozwala bezproblemowo obsłużyć każdego pasażera indywidualnie. Kluczowe jednak staje się pozycjonowanie windy w okresach braku ruchu.

5.4. Algorytmy

Do stworzonego środowiska wprowadzany był agent *RL*, który wchodził z nim w interakcję opisaną w sekcji 3.1.2. Korzystając ze zbieranego doświadczenia uczył się i przybliżał do optymalnej strategii zgodnie z wybranym algorytmem nauki. Algorytm ten, wraz z opisującymi go parametrami, definiują proces nauki agenta. Wykorzystanymi w tej pracy algorytmami są opisany w sekcji 3.2.2.1 *Q-Learning* oraz pochodne mu algorytmy wraz z rozszerzeniami. Dokładna ich lista jest następująca:

1. *Q-Learning*,
2. *Q-Learning* z buforem pamięci,
3. *Deep Q-Learning* z buforem pamięci,
4. *Double Deep Q-Learning* z buforem pamięci,
5. *Double Deep Q-Learning* z priorytetowym buforem pamięci,
6. *Double Deep Q-Learning* z pałacem pamięci.

Dodatkowo wszystkie z nich korzystały z algorytmu *ε-greedy* zapewniającego zbieżność do optymalnej funkcji Q^* poprzez ciągłe niezerowe prawdopodobieństwo eksploracji.

6. Szczegóły implementacyjne

W tym rozdziale opisana została techniczna strona pracy. Wymienione zostały wykorzystane w projekcie technologie i biblioteki oraz opisana jego struktura plików.

6.1. Wykorzystane technologie

W pracy wykorzystano środowisko bazujące na języku *Python*, ponieważ oferuje on wiele szeroko rozwiniętych bibliotek dedykowanych do przetwarzania danych i algorytmów sztucznej inteligencji. Wykorzystana została wersja 3.9.7 wraz z następującymi pakietami:

- *PyTorch* – wysoce zoptymalizowana i rozwinięta biblioteka do obliczeń na tensorach oraz do implementacji technik *Machine Learning* i *Deep Learning*, wspierająca obliczenia na układach *GPU*;

- *NumPy* – szeroko używana i wyspecjalizowana biblioteka naukowa do obliczeń matematycznych i przekształceń algebraicznych na wielowymiarowych macierzach;

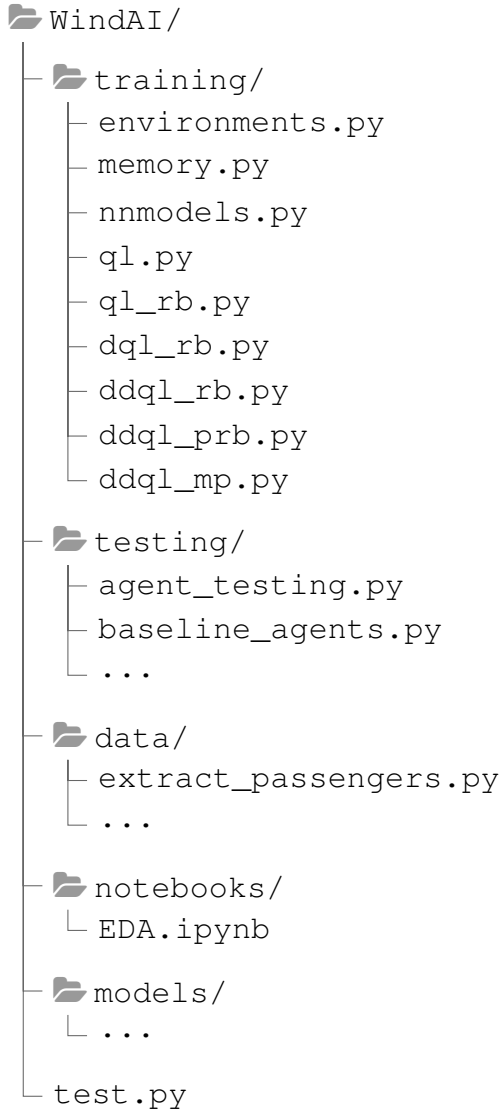
- *pandas* – biblioteka wykorzystywana do operacji na tabelarycznych strukturach danych oraz szeregach czasowych;

- *matplotlib* oraz *seaborn* – biblioteki do tworzenia wykresów i wizualizacji danych statystycznych;

- *Pygame* – biblioteka do implementacji interfejsów graficznych gier.

6.2. Struktura kodu

Struktura projektu prezentuje się następująco:



Pakiet `training` zawiera moduły i skrypty wykorzystane w procesie treningu. Składają się na nie: moduł `environments` zawierający zaimplementowane środowiska *RL*, moduł `memory` zawierający implementacje struktur pamięci wykorzystywanych w technikach *Experience Replay*, moduł `nnmodels` z klasami sieci neuronowych wykorzystywanych do modelowania funkcji Q w metodach *Deep Q-Learning* oraz 6 skryptów implementacji algorytmów uczenia ze wzmocnieniem: *Q-Learning*, *Q-Learning* z buforem pamięci, *Deep Q-Learning* z buforem pamięci, *Double Deep Q-Learning* z buforem pamięci, *Double Deep Q-Learning* z priorytetowym buforem pamięci, *Double Deep Q-Learning* z pałacem pamięci.

Pakiet `testing` zawiera moduły wykorzystane w trybie testowania wytrenowanych agentów. Moduł `agent_testing` zawiera klasy i funkcje potrzebne do procesu wgrywania zserializowanych agentów, uruchamiania symulacji w odpowiednim środowisku oraz zbierania metryk. Moduł `baseline_agents` zawiera implementacje algorytmów wzorcowych, czyli

heurystycznych, wykorzystywanych do oceny osiągów wytrenowanych agentów. Dodatkowo zawiera moduły przygotowanych testów uruchamianych z poziomu głównego folderu przy pomocy skryptu `test`.

Folder `data` zawiera skrypt `extract_passengers` odpowiedzialny za pobranie danych z budynku ASTOR, wczytanie i zliczenie pasażerów i zapisanie odpowiednich macierzy częstotliwości pojawiania się pasażerów w plikach w tym samym folderze.

Folder `notebooks` zawiera notatnik *Jupyter'a* użyty do przeprowadzenia wstępnego badania eksploracyjnego danych pobranych z ASTOR (ang. *Exploratory Data Analysis, EDA*).

Folder `models` zawiera zapisanych wytrenowanych agentów, pogrupowanych w foldery według typu środowiska, w którym działają. Każdy agent posiada osobny podfolder, w którego plikach znajdują się dokładne ustawienia dotyczące procesu treningu, środowiska, algorytmu, struktury oraz informacje o przebiegu nauki wraz z zapisanymi punktami kontrolnymi. Tak usystematyzowana struktura pozwala w jednoznaczny sposób ustalić parametry agenta potrzebne w testach.

7. Nauka agentów

W trakcie badań uruchomione zostały liczne procesy nauki dla zróżnicowanych par środowiska i agenta. Były one z każdą iteracją bardziej rozwinięte i korzystały z wniosków zebranych w poprzednich stadiach projektu. W tym rozdziale przedstawione zostaną, w kolejności implementacyjnej, wybrane i zasługujące na uwagę procesy nauki.

7.1. Pojedynczy pasażer

Najmniej skomplikowane środowisko utworzone w tej pracy bazuje na najprostszym rozkładzie zakładającym pojawianie się tylko jednego pasażera na początku symulacji (patrz punkt (1) sekcja 5.2). Środowisko to posiada również skonstruowany zbiór Ω, do którego należy każdy stan po obsłużeniu pasażera. Zatrzymanie środowiska jest zatem kojarzone z nagrodą za osiągnięcie stanu docelowego. Pozostałe parametry środowiska przedstawia tabela 7.1.

Parametr		**Wartość**	**Uwagi**
Liczba pięter		20	
Pojemność kabiny		1	Wystarczające dla jednego pasażera
Pojemność piętra		1	
Konstrukcja wektora stanu x_t		1. Pozycja windy, stan przycisków	Przekazuje wszystkie potrzebne informacje, generuje 40 unikalnych stanów
Rozkład pojawiania się pasażerów		1. Pojedynczy pasażer	
Współczynniki funkcji nagrody R	s_c	1	Kara za ruch windy
	s_d	100	Kara za nielegalny ruch
	s_f	100	Nagroda za zatrzymanie środowiska wynikające z obsłużenia pasażera
	reszta	0	

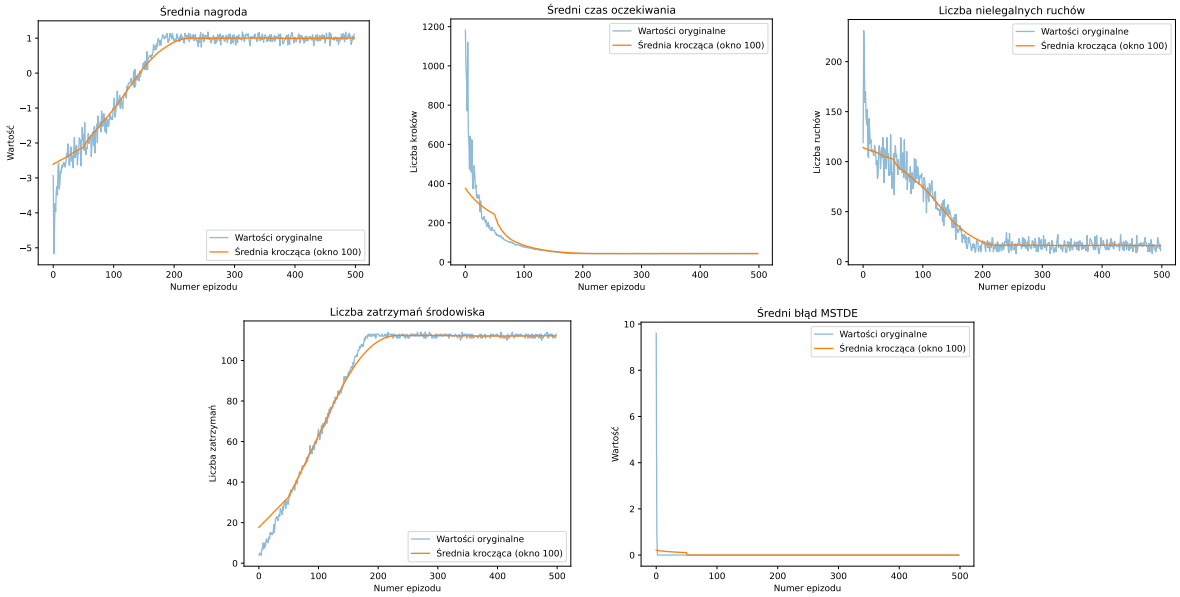
Tabela 7.1. Parametry środowiska z pojedynczym pasażerem

Agent wprowadzony do opisanego środowiska wykorzystywał do nauki algorytm Q-*Learning* z buforem pamięci oraz strategią ε-*greedy*. Trening był rozdzielony na epizody o równej liczbie kroków symulacji. Parametry algorytmu opisuje tabela 7.2.

W ramach każdego epizodu zbierane były wartości metryk: średniej nagrody na jeden krok, średniego czasu oczekiwania obsłużonych pasażerów (czas przebywania w systemie), liczbę nielegalnych ruchów, liczbę zatrzymań środowiska oraz średniej wartości kwadratu różnicy chwilowej (*Mean Squared Temporal Difference Error*). Przebieg ich wartości w trakcie nauki przedstawiają wykresy na rysunku 7.1.

Parametr	Wartość	Opis
Liczba epizodów	500	
Liczba kroków na epizod	5000	
Stała ucząca α	0.5	Ustala prędkość uczenia (ang. *learning rate*)
Współczynnik γ	0.99	Wartościuje nagrody natychmiastowe do przyszłych (ang. *discount rate*)
Początkowa wartość ε	1	Prawdopodobieństwo eksploracji na początku nauki
Końcowa wartość ε	0.1	Wartość, po której osiągnięciu ε nie maleje
Krok zmniejszania ε	0.005	Wartość liniowego zmniejszania ε co epizod nauki
Minimalna wielkość bufora pamięci	500	Liczba krotek doświadczenia, po których zebraniu rozpoczyna się proces nauki
Maksymalna wielkość bufora pamięci	2000	Liczba ostatnich krotek przechowywanych w buforze
Okres pobierania próbki z bufora mierzony liczbą kroków	8	Liczba kroków co które pobiera się próbkę z bufora
Wielkość próbki	32	Liczba losowych kotek pobieranych z bufora

Tabela 7.2. Parametry algorytmu *Q-Learning* z buforem pamięci i strategią *ε-greedy* wykorzystanego przez agenta, który uczył się w środowisku z pojedynczym pasażerem.



Rys. 7.1. Wykresy metryk zbieranych w trakcie nauki agenta w środowisku z pojedynczym pasażerem. Wartość średniej nagrody rośnie, to samo dotyczy liczby zatrzymań środowiska - czyli liczby obsłużeń pasażera. Wartości średniego czasu oczekiwania, liczby nielegalnych ruchów oraz średniego błędu maleją. Po osiągnięciu okolic ekstremum widoczny szum jest wynikiem losowej eksploracji wprowadzanej przez strategię *ε-greedy*.

Źródło: opracowanie własne

7.2. Nieostrożny wybór nagrody

Kolejne stworzone środowisko zakładało system siedmiopiętrowy. Zdecydowano się na równomierny rozkład pasażerów względem trasy, z oczekiwanym okresem pojawiania się równym 5 krokom czyniącym z niego środowisko wymagające. Jednak wybrana nagroda, bazująca na różnicy w wartości sumy czasu oczekiwania pasażerów, okazała się nieadekwatna w kontekście docelowych metryk. Jest to przykład nieostrożnie wybranej konstrukcji funkcji nagrody. Parametry środowiska przedstawia tabela 7.3.

Parametr		**Wartość**	**Uwagi**
Liczba pięter		7	
Pojemność kabiny		∞	Wprowadzenie uproszczenia względem rzeczywistego systemu
Pojemność piętra		1	
Konstrukcja wektora stanu x_t		1. Pozycja windy, stan przycisków	Dla wielu pasażerów jest to minimalna ilość potrzebnych informacji, generuje $7 \cdot 2^{14} = 114688$ unikalnych stanów
Rozkład pojawiania się pasażerów		3. Równomierny ruch	Oczekiwana częstotliwość pojawienia się dowolnego pasażera jest równa $\frac{1}{5}$
Współczynniki funkcji nagrody R	s_c	1	Kara za ruch windy
	s_d	10^9	Kara za nielegalny ruch
	s_g	1	Nagroda za różnicę w sumie oczekiwania pasażerów
	reszta	0	

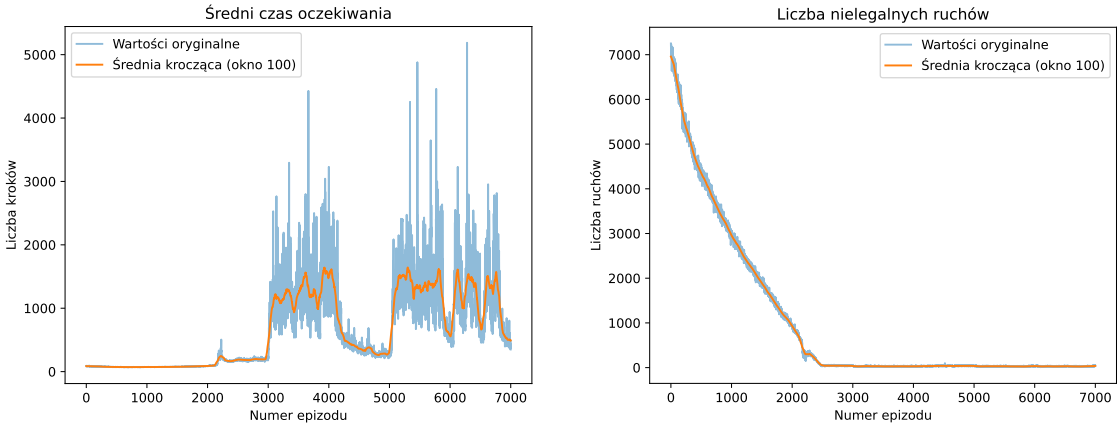
Tabela 7.3. Parametry środowiska z z nieostrożnie wybraną nagrodą

Agent wprowadzony do opisanego środowiska korzystał z algorytmu *Q-Learning* z buforem pamięci i strategią *ε-greedy*. Jego dokładne parametry opisuje tabela 7.4.

Parametr	**Wartość**	**Opis**
Liczba epizodów	7000	Bardziej skomplikowane środowisko wymaga dłuższej nauki
Liczba kroków na epizod	75000	
Stała ucząca α	0.5	Ustala prędkość uczenia (ang. *learning rate*)
Współczynnik γ	0.99	Wartościuje nagrody natychmiastowe do przyszłych (ang. *discount rate*)
Początkowa wartość ε	1	Prawdopodobieństwo eksploracji na początku nauki
Końcowa wartość ε	0.1	Wartość, po której osiągnięciu ε nie maleje
Krok zmniejszania ε	0.004	Wartość liniowego zmniejszania ε co epizod nauki
Minimalna wielkość bufora pamięci	500	Liczba krotek doświadczenia, po których zebraniu rozpoczyna się proces nauki
Maksymalna wielkość bufora pamięci	150000	Liczba ostatnich krotek przechowywanych w buforze, równa 2 ostatnim epizodom
Okres pobierania próbki z bufora mierzony liczbą kroków	10	Liczba kroków co które pobiera się próbkę z bufora
Wielkość próbki	50	Liczba losowych kotek pobieranych z bufora

Tabela 7.4. Parametry algorytmu *Q-Learning* z buforem pamięci i strategią *ε-greedy* wykorzystanego przez agenta umieszczonego w środowisku z z nieostrożnie wybraną nagrodą

W każdym epizodzie nauki zbierano wartości liczby nielegalnych ruchów oraz średniego czasu oczekiwania obsłużonych pasażerów. Wykresy przedstawiające ich przebieg zawarto na rysunku 7.2.



Rys. 7.2. Wykresy metryk zbieranych w trakcie nauki agenta w środowisku z nieostrożnie wybraną nagrodą. Pomimo, że wartość nagrody z epizodu na epizod rosła (przesłanką czego jest malejąca liczba nielegalnych ruchów), to docelowa metryka długości czasu oczekiwania nie zmieniała się zgodnie z założeniem.

Źródło: opracowanie własne

7.3. Rozkład z budynku ASTOR według trasy

Następne środowisko korzysta z rozkładu pojawiania się pasażerów zamodelowanego na podstawie rzeczywistego systemu w budynku ASTOR. Oczekiwany okres pojawiania się pasażerów w wykorzystanym rozkładzie to około 5 kroków, co dla siedmiopiętrowego systemu generuje ruch wymagający. Nagroda bazowała na ilości wciśniętych przycisków, co może być interpretowane jako swego rodzaju temperatura systemu - im więcej pasażerów w systemie, tym mniejsza nagroda. Agent zatem będzie starał się minimalizować ich liczbę poprzez odpowiednie rozprowadzanie. Parametry środowiska ilustruje tabela 7.5.

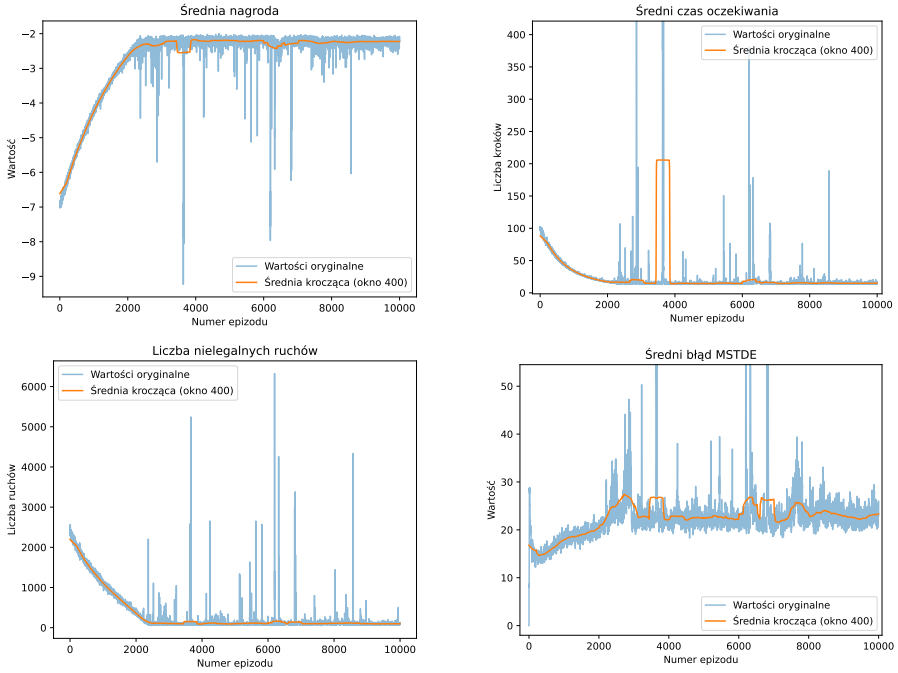
Parametr		**Wartość**	**Uwagi**
Liczba pięter		7	Odpowiadają piętrom od -1 do 5 w budynku ASTOR
Pojemność kabiny		∞	Uproszczenie względem rzeczywistego systemu
Pojemność piętra		1	
Konstrukcja wektora stanu x_t		1. Pozycja windy, stan przycisków	Minimalna ilość potrzebnych informacji, generuje $7 \cdot 2^{14} = 114688$ unikalnych stanów
Rozkład pojawiania się pasażerów		4. Rozkład z budynku ASTOR według trasy	Wartości z macierzy $\mathbb{T}$ przemnożono 16-krotnie otrzymując oczekiwaną częstotliwość równą około $\frac{1}{5}$
Współczynniki funkcji nagrody R	s_a	1	Kara za każdy wciśnięty przycisk (bez rozróżnienia na góra/dół)
	s_d	20	Kara za nielegalny ruch
	pozostałe	0	

Tabela 7.5. Parametry środowiska z rozkładem z ASTOR według trasy

W opisanym środowisku zdecydowano się umieścić agenta wykorzystującego technikę *Double Deep Q-Learning* z buforem pamięci oraz strategią ε*-greedy* w celu zapobiegania niestabilności nauki wynikającej z wariancji kolejnych obserwacji. Dokładne parametry algorytmu umieszczono w tabeli 7.6. Wykresy zebranych w trakcie nauki metryk umieszczono na rysunku 7.3.

Parametr	**Wartość**	**Opis**
Liczba epizodów	10000	Długośc nauki
Liczba kroków na epizod	25000	
Współczynnik γ	0.99	Wartościuje nagrody natychmiastowe do przyszłych (ang. *discount rate*)
Początkowa wartość ε	1	
Końcowa wartość ε	0.1	Wartość ε co epizod nauki maleje liniowo o ustaloną wartość, aż do osiągnięcia wartości minimalnej, na której się zatrzymuje
Krok zmniejszania ε	0.004	
Minimalna wielkość bufora pamięci	50000	Liczba krotek doświadczenia, po których zebraniu rozpoczyna się proces nauki
Maksymalna wielkość bufora pamięci	125000	Liczba ostatnich krotek przechowywanych w buforze, równa 5 ostatnim epizodom
Okres pobierania próbki z bufora mierzony liczbą kroków	10	Liczba kroków co które pobiera się próbkę z bufora
Wielkość próbki	32	Liczba losowych kotek pobieranych z bufora
Interwał kopiowania	10000	Liczba kroków co które następuje kopiowanie wag pomiędzy sieciami w wariancie *Double Q-Learning*
Architektura sieci	*feedforward*	Sieć modelująca funkcję Q w wariancie *Deep Q-Learning*
	15	Długość wektora stanu
Wielkości warstw w sieci	128, 32, 8	Trzy warstwy ukryte
	3	Liczba możliwych akcji
Algorytm nauki sieci	*RMSProp*	Algorytm odpowiedzialny za aktualizację wag sieci, będący rozszerzeniem *SGD* o koncept pędu
Funkcja błędu	*MSTE*	Funkcja której wartość jest minimalizowana przez sieć, *Mean Squared TD Error*

Tabela 7.6. Parametry algorytmu *Double Deep Q-Learning* z buforem pamięci i strategią ε-*greedy* wykorzystanego przez agenta w środowisku z rozkładem z ASTOR według trasy



Rys. 7.3. Wykresy metryk zbieranych w trakcie nauki agenta w środowisku z rozkładem z ASTOR według trasy. Wartości średniej nagrody wykazują tendencję rosnącą, natomiast liczba nielegalnych ruchów oraz średni czas oczekiwania malejącą (co pokazują wykresy średniej kroczącej). Średni błąd nie spada do zera ponieważ środowisko jest obarczone losowością. Widoczne zaburzenia wynikają z chaotyczności środowiska oraz jego odkrywaniu przez agenta.

Źródło: opracowanie własne

7.4. Stan z rozróżnieniem przycisków w górę i w dół

Kolejne trzy pary zawierały jednakowe środowisko, co pozwoliło na bezpośrednie porównanie algorytmów nauki wykorzystanych przez agentów. Środowisko wykorzystywało rozkład pojawiania się pasażerów zależny od obieranej trasy, który zamodelowano na podstawie systemu w budynku ASTOR. W tej instancji oczekiwany okres pojawiania się pasażerów to około 10 kroków. Rozszerzeniem w stosunku do poprzedniego środowiska była również reprezentacja wektora stanu zawierająca podział na przyciski góry i dołu na piętrach wprowadzające dodatkową informację. Parametry środowiska ukazuje tabela 7.7.

Parametr		**Wartość**	**Uwagi**
Liczba pięter		7	Odpowiadają piętrom od -1 do 5 w budynku ASTOR
Pojemność kabiny		∞	Uproszczenie względem rzeczywistego systemu
Pojemność piętra		1	
Konstrukcja wektora stanu x_t		2. Pozycja windy, stan przycisków góra/dół	Rozróżnienie na przyciski wezwania do góry i w dół na piętrach, generuje $7 \cdot 2^{7+6+6} \approx 3.7 \cdot 10^6$ unikalnych stanów
Rozkład pojawiania się pasażerów		4. Rozkład z budynku ASTOR według trasy	Oczekiwana częstotliwość równa $\frac{1}{10}$
Współczynniki funkcji nagrody R	s_a	0.1	Kara za każdy wciśnięty przycisk (z rozróżnieniem na góra i dół)
	s_d	2	Kara za nielegalny ruch
	pozostałe	0	

Tabela 7.7. Parametry środowiska z rozróżnieniem na przyciski wezwania na piętrach na w górę i w dół.

7.4.1. *DDQL* z buforem pamięci

Pierwszy agent uruchomiony w opisanym środowisku wykorzystywał algorytm *Double Deep Q-Learning* z buforem pamięci. Parametry algorytmu zawiera tabela 7.8.

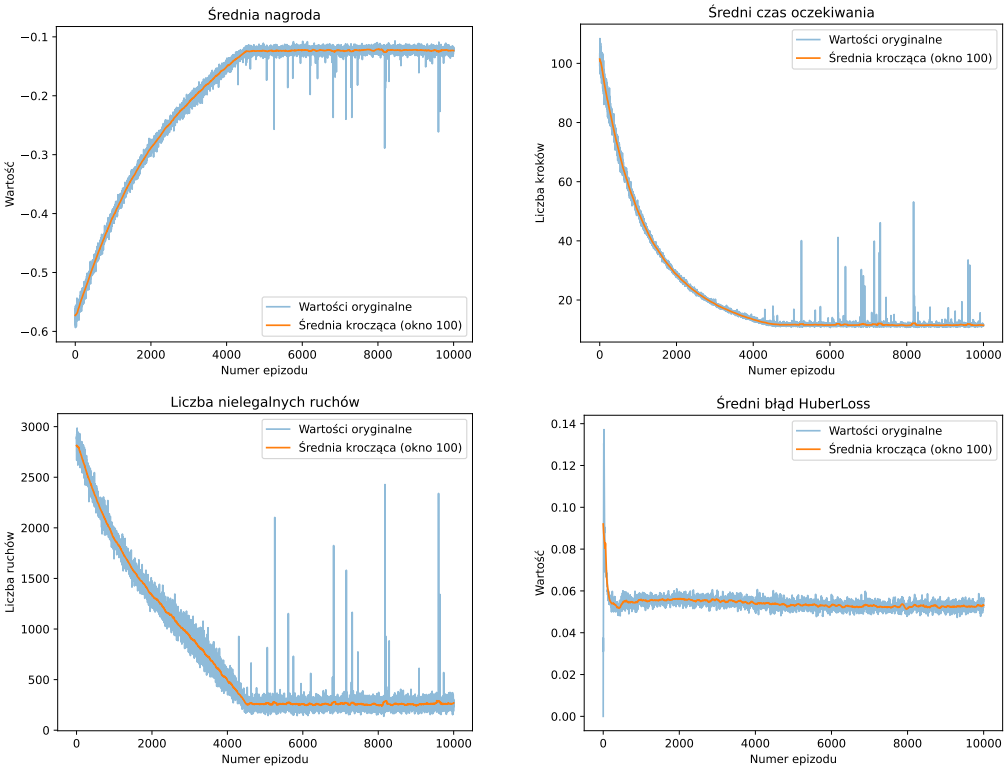
W trakcie nauki zbierano wartości średniej nagrody, średniego czasu oczekiwania obsłużonych pasażerów, średniej wartości błędu oraz liczby nielegalnych ruchów, których wykresy umieszczono na rysunku 7.4.

7.4.2. *DDQL* z pałacem pamięci

Drugi agent wprowadzony do opisanego środowiska wykorzystywał algorytm *Double Deep Q-Learning* z pałacem pamięci. Podział na pokoje następował względem numeru piętra, na którym znajduje się aktualnie winda. Pobierane do nauki krotki doświadczenia były tym samym równomiernie rozłożone względem tej wartości. Parametry algorytmu zawiera tabela 7.9. Wykresy zebranych w trakcie nauki metryk umieszczono na rysunku 7.5.

Parametr	Wartość	Opis
Liczba epizodów	10000	Długosc nauki
Liczba kroków na epizod	30000	
Współczynnik γ	0.99	Wartościuje nagrody natychmiastowe do przyszłych (ang. *discount rate*)
Początkowa wartość ε	1	
Końcowa wartość ε	0.1	Wartość ε co epizod nauki maleje liniowo o ustaloną wartość, aż do osiągnięcia wartości minimalnej, na której się zatrzymuje
Krok zmniejszania ε	0.0002	
Minimalna wielkość bufora pamięci	50000	Liczba krotek doświadczenia, po których zebraniu rozpoczyna się proces nauki
Maksymalna wielkość bufora pamięci	150000	Liczba ostatnich krotek przechowywanych w buforze, równa 5 ostatnim epizodom
Okres pobierania próbki z bufora mierzony liczbą kroków	8	Liczba kroków co które pobiera się próbkę z bufora
Wielkość próbki	32	Liczba losowych kotek pobieranych z bufora
Interwał kopiowania	10000	Liczba kroków co które następuje kopiowanie wag pomiędzy sieciami w wariancie *Double Q-Learning*
Architektura sieci	*feedforward*	Sieć modelująca funkcję Q w wariancie *Deep Q-Learning*
	20	Długość wektora stanu
Wielkości kolejnych warstw sieci	128	Jedna warstwa ukryta
	3	Liczba możliwych akcji
Algorytm nauki sieci	*Adam*	Algorytm odpowiedzialny za aktualizację wag sieci, będący rozszerzeniem *SGD*
Funkcja błędu	*Huber Loss*	Funkcja której wartość jest minimalizowana przez sieć, mniej czuła na wartości odstające niż *MSE*

Tabela 7.8. Parametry algorytmu *Double Deep Q-Learning* z buforem pamięci i strategią ε*-greedy* wykorzystanego przez agenta w środowisku z przyciskami góra/dół

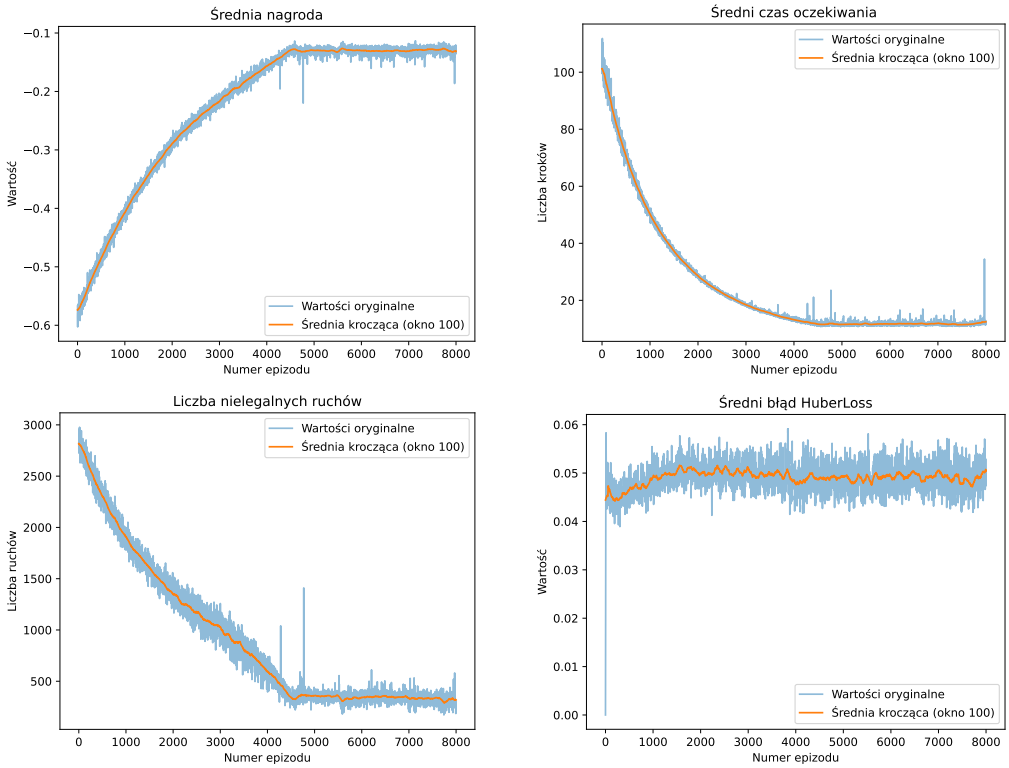


Rys. 7.4. Wykresy metryk zbieranych w trakcie nauki agenta wykorzystującego bufor pamięci. Wartości średniej nagrody wykazują tendencję rosnącą, natomiast liczba nielegalnych ruchów oraz średni czas oczekiwania malejącą. Średni błąd nie spada do zera ponieważ środowisko jest obarczone losowością.

Źródło: opracowanie własne

Parametr	Wartość	Opis
Liczba epizodów	8000	Długosc nauki
Liczba kroków na epizod	30000	
Współczynnik γ	0.99	Wartościuje nagrody natychmiastowe do przyszłych (ang. *discount rate*)
Początkowa wartość ε	1	
Końcowa wartość ε	0.1	Wartość ε co epizod nauki maleje liniowo o ustaloną wartość, aż do osiągnięcia wartości minimalnej, na której się zatrzymuje
Krok zmniejszania ε	0.0002	
Minimalna wielkość pałacu pamięci	50000	Liczba krotek doświadczenia, po których zebraniu rozpoczyna się proces nauki
Maksymalna wielkość pałacu pamięci	150000	Liczba ostatnich krotek przechowywanych w pałacu, równa 5 ostatnim epizodom (rozłożona równo na 7 pokoi)
Okres pobierania próbki z bufora mierzony liczbą kroków	8	Liczba kroków co które pobiera się próbkę z pałacu
Wielkość próbki	28	Liczba losowych kotek pobieranych z pałacu (równomiernie rozłożona względem 7 pięter)
Interwał kopiowania	10000	Liczba kroków co które następuje kopiowanie wag pomiędzy sieciami w wariancie *Double Q-Learning*
Architektura sieci	*feedforward*	Sieć modelująca funkcję Q w wariancie *Deep Q-Learning*
	20	Długość wektora stanu
Wielkości kolejnych warstw sieci	128, 32, 8	Trzy warstwy ukryte
	3	Liczba możliwych akcji
Algorytm nauki sieci	*Adam*	Algorytm odpowiedzialny za aktualizację wag sieci, będący rozszerzeniem *SGD*
Funkcja błędu	*Huber Loss*	Funkcja której wartość jest minimalizowana przez sieć, mniej czuła na wartości odstające niż *MSE*

Tabela 7.9. Parametry algorytmu *Double Deep Q-Learning* z pałacem pamięci i strategią ε-*greedy* wykorzystanego przez agenta w środowisku z przyciskami góra/dół



Rys. 7.5. Wykresy metryk zbieranych w trakcie nauki agenta wykorzystującego pałac pamięci. Przebieg nauki jest płynniejszy niż dla agenta wykorzystującego bufor pamięci.

Źródło: opracowanie własne

7.4.3. *DDQL* z priorytetowym buforem pamięci

Ostatni z agentów umieszczonych w środowisku opisanym w tabeli 7.7 wykorzystywał algorytm *Double Deep Q-Learning* z priorytetowym buforem pamięci. Priorytetowy bufor pamięci przyspiesza proces nauki dzięki łączeniu prawdopodobieństwa wylosowania danej krotki z bufora z jej przydatnością. Do określenia przydatności wykorzystano metrykę chwilowej różnicy *TD*. W ten sposób przetestowano wykorzystanie trzech różnych technik *Experience Replay* dla tego samego środowiska, co umożliwiło ich porównanie. Parametry algorytmu przedstawiono w tabeli 7.10. Wykresy zebranych w trakcie nauki metryk umieszczono na rysunku 7.6.

7.5. Rozkład z budynku ASTOR według godziny i trasy

Kolejne stworzone środowisko wprowadza dodatkowy wymiar stanu jakim jest aktualna godzina. Rozkład pasażerów bazował na macierzy częstotliwości pojawiania się pasażerów w budynku ASTOR z rozróżnieniem na obieraną trasę oraz aktualną godzinę. Ruch w budynku jest zmienny względem godziny i wprowadzenie jej do konstrukcji środowiska czyni je bardziej złożonym. Średni okres pojawiania się pasażerów waha się od kilkunastu do kilkudziesięciu kroków w trakcie dnia. Reagując na tę zmienność agent może wypracować osobną strategię dla każdej godziny. Parametry środowiska zawarto w tabeli 7.11.

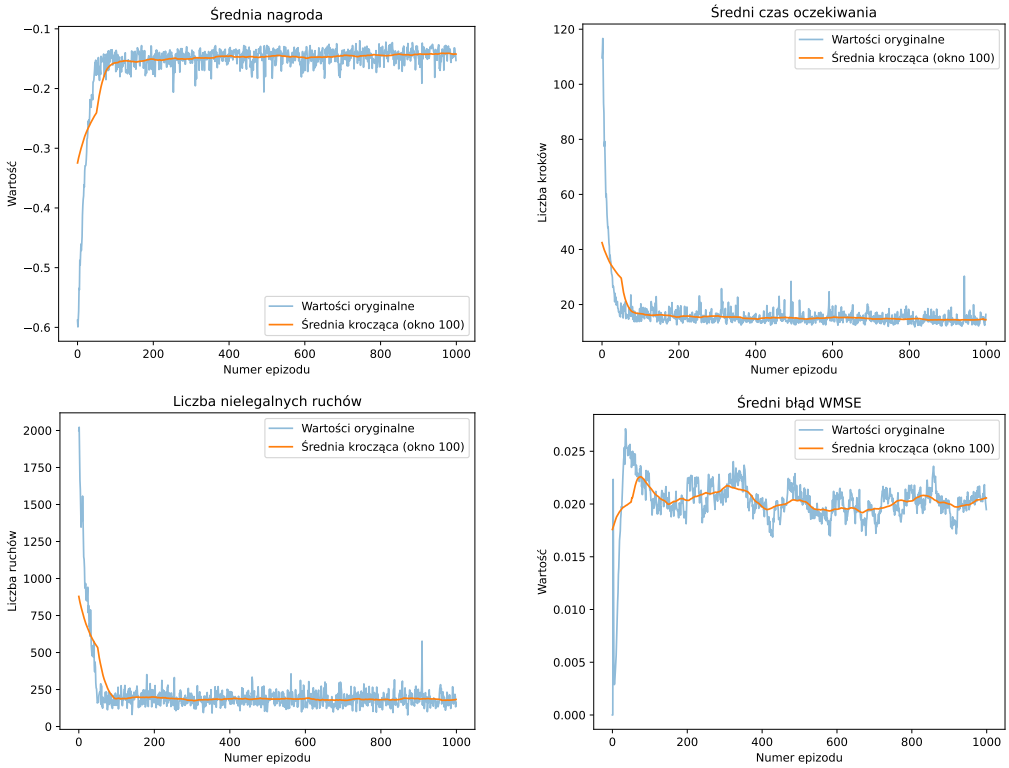
Parametr		Wartość	Uwagi
Liczba pięter		7	Odpowiadają piętrom od -1 do 5 w budynku ASTOR
Pojemność kabiny		∞	Uproszczenie względem rzeczywistego systemu
Pojemność piętra		1	
Konstrukcja wektora stanu x_t		3. Pozycja windy, godzina w symulacji, stan przycisków góra/dół	Rozróżnienie na przyciski wezwania do góry i w dół na piętrach, generuje $16 \cdot 7 \cdot 2^{19} \approx 5.9 \cdot 10^7$ unikalnych stanów
Rozkład pojawiania się pasażerów		5. Rozkład z budynku ASTOR według godziny i trasy	Średnia częstotliwość według godziny równa $\frac{1}{80}$
Godziny		$7, 8, 9 \ldots 21, 22$	Możliwe do ustawienia godziny w środowisku (niewykorzystane wykazywały zbyt mały ruch)
Współczynniki funkcji nagrody R	s_a	0.1	Kara za każdy wciśnięty przycisk (z rozróżnieniem na góra i dół)
	s_d	2	Kara za nielegalny ruch
	pozostałe	0	

Tabela 7.11. Parametry środowiska zależnego od godziny

Wprowadzony do opisanego środowiska agent wykorzystywał algorytm *Double Deep Q-Learning* z buforem pamięci. Jego parametry opisuje tabela 7.12. Wykresy zebranych w trakcie nauki metryk umieszczono na rysunku 7.7.

Parametr	Wartość	Opis
Liczba epizodów	1000	Długosc nauki
Liczba kroków na epizod	20000	
Współczynnik γ	0.99	Wartościuje nagrody natychmiastowe do przyszłych (ang. *discount rate*)
Początkowa wartość ε	1	
Końcowa wartość ε	0.1	Wartość ε co epizod nauki maleje liniowo o ustaloną wartość, aż do osiągnięcia wartości minimalnej, na której się zatrzymuje
Krok zmniejszania ε	0.018	
Minimalna wielkość bufora pamięci	50000	Liczba krotek doświadczenia, po których zebraniu rozpoczyna się proces nauki
Maksymalna wielkość bufora pamięci	1000000	Liczba ostatnich krotek przechowywanych w buforze, równa 50 ostatnim epizodom
Okres pobierania próbki z bufora mierzony liczbą kroków	4	Liczba kroków co które pobiera się próbkę z bufora
Wielkość próbki	32	Liczba losowych kotek pobieranych z bufora
Interwał kopiowania	40000	Liczba kroków, co które następuje kopiowanie wag pomiędzy sieciami w wariancie *Double Q-Learning*
Architektura sieci	*feedforward*	Sieć modelująca funkcję Q w wariancie *Deep Q-Learning*
	20	Długość wektora stanu
Wielkości kolejnych warstw sieci	128	Warstwa ukryta
	3	Liczba możliwych akcji
Algorytm nauki sieci	*Adam*	Algorytm odpowiedzialny za aktualizację wag sieci, jest rozszerzeniem *SGD*
Funkcja błędu	*WMSE*	Funkcja minimalizowana przez sieć, ważony błąd średniokwadratowy *Weighted MSE*

Tabela 7.10. Parametry algorytmu *Double Deep Q-Learning* z priorytetowym buforem pamięci i strategią ε-*greedy* wykorzystanego przez agenta w środowisku z przyciskami góra/dół

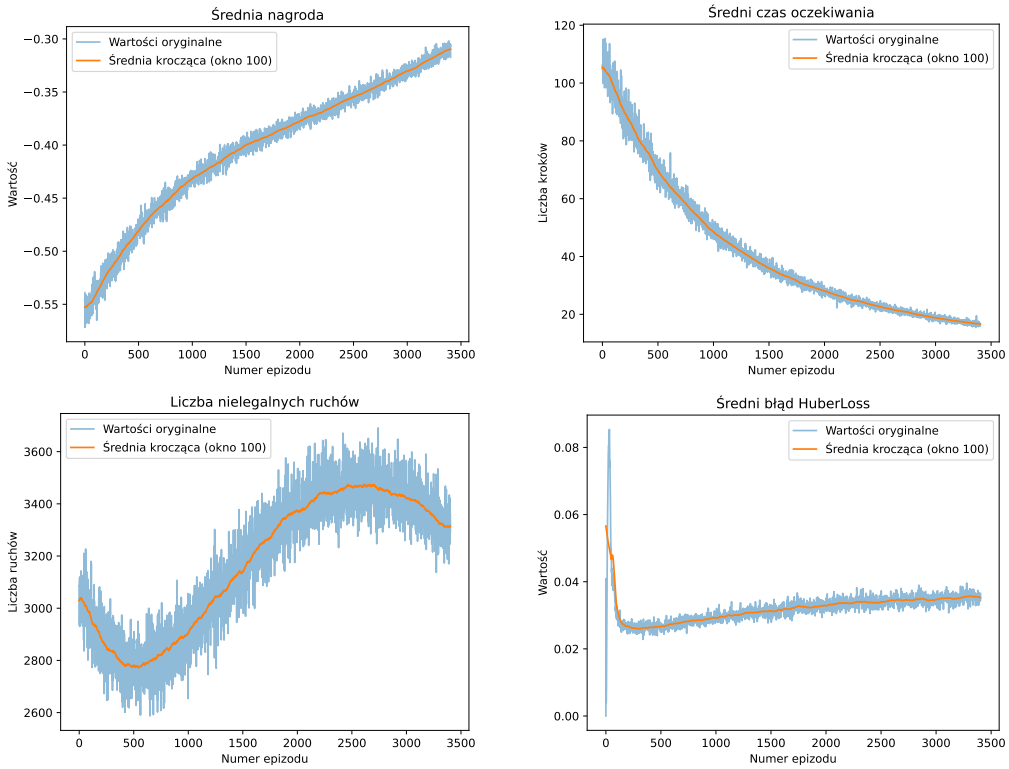


Rys. 7.6. Wykresy metryk zbieranych w trakcie nauki agenta wykorzystującego priorytetowy bufor pamięci. Przebieg nauki jest znacząco krótszy niż dla agentów wykorzystującego bufor pamięci i pałac pamięci.

Źródło: opracowanie własne

Parametr	Wartość	Opis
Liczba epizodów	3404	Długośc nauki, nieplanowanie przerwanej
Liczba kroków na epizod	32000	Dla każdej z 16 godzin przypadało 2000 kroków, po których godzina środowiska była zmieniana
Współczynnik γ	0.99	Wartościuje nagrody natychmiastowe do przyszłych (ang. *discount rate*)
Początkowa wartość ε	1	
Końcowa wartość ε	0.1	Wartość ε co epizod nauki maleje liniowo o ustaloną wartość, aż do osiągnięcia wartości minimalnej, na której się zatrzymuje
Krok zmniejszania ε	0.00015	
Minimalna wielkość bufora pamięci	50000	Liczba krotek doświadczenia, po których zebraniu rozpoczyna się proces nauki
Maksymalna wielkość bufora pamięci	150000	Liczba ostatnich krotek przechowywanych w buforze, równa około 5 ostatnim epizodom
Okres pobierania próbki z bufora mierzony liczbą kroków	8	Liczba kroków co które pobiera się próbkę z pałacu
Wielkość próbki	32	Liczba losowych kotek pobieranych z bufora
Interwał kopiowania	10000	Liczba kroków, co które następuje kopiowanie wag pomiędzy sieciami w wariancie *Double Q-Learning*
Architektura sieci	*feedforward*	Sieć modelująca funkcję Q w wariancie *Deep Q-Learning*
	21	Długość wektora stanu
Wielkości kolejnych warstw sieci	128, 64, 8	Trzy warstwy ukryte
	3	Liczba możliwych akcji
Algorytm nauki sieci	*Adam*	Algorytm odpowiedzialny za aktualizację wag sieci, jest rozszerzeniem *SGD*
Funkcja błędu	*HuberLoss*	Funkcja minimalizowana przez sieć, bardziej odporny na wartości odstające niż *MSE*

Tabela 7.12. Parametry algorytmu *DDQL* z buforem pamięci i strategią *ε-greedy* wykorzystanym przez agenta w środowisku zależnym od godziny



Rys. 7.7. Wykresy metryk zbieranych w trakcie nauki agenta w środowisku zależnym od godziny. Nauka została przerwana przed osiągnięciem przez ε wartości końcowej. Ma to widoczny efekt w wykresach metryk, które mimo zadowalających trendów, nie osiągnęły jeszcze stanu *plateau*.

Źródło: opracowanie własne

7.6. Stan z wymiarem czasu aktywności przycisków

Ostatnie prezentowane środowisko jest znacznym rozwinięciem opisanego w podsekcji 7.4. Wprowadza bowiem wymiar czasu w odniesieniu do wciśniętych przycisków. Stan środowiska zawiera wówczas nieporównywalnie więcej informacji w stosunku do uwzględniającego jedynie logiczną fazę przycisków. Jednocześnie prowadzi to jednak do konstrukcji nieskończonego zbioru stanów. Aby przywrócić jego policzalność, zdecydowano się wprowadzić warunek zatrzymania środowiska, jeśli istnieje pasażer oczekujący w środowisku dłużej niż 50 kroków. Zatrzymanie w tym kontekście jest traktowane negatywnie i adekwatnie karane. Dzięki takiemu zabiegowi przestrzeń stanów zawiera "jedynie" $7 \cdot 4 \cdot 50^{20} \approx 2.67 \cdot 10^{35}$ elementów (gdzie czynnik 4 wynika z liczby wybranych godzin wpływających na rozkład pasażerów). Dokładne parametry środowiska opisuje tabela 7.13.

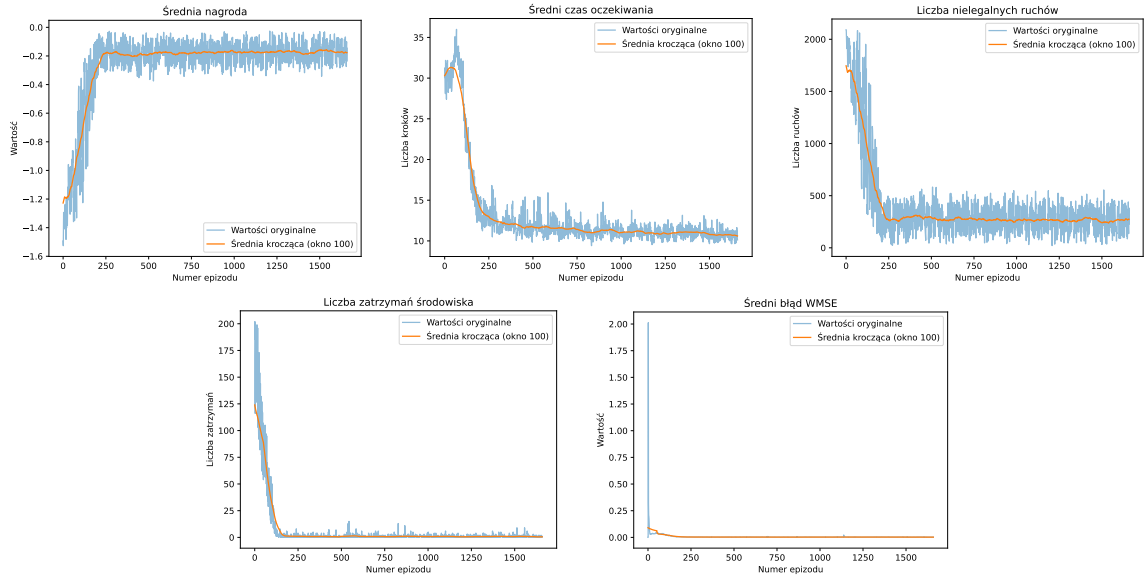
Parametr		**Wartość**	**Uwagi**
Liczba pięter		7	Odpowiadają piętrom od -1 do 5 w budynku ASTOR
Pojemność kabiny		∞	Uproszczenie względem rzeczywistego systemu
Pojemność piętra		1	
Konstrukcja wektora stanu x_t		4. Pozycja windy, godzina, czas aktywności przycisków góra/dół	Najbardziej rozwinięta konstrukcja wektora stanu, generuje $\approx 2.67 \cdot 10^{35}$ unikalnych stanów
Rozkład pojawiania się pasażerów		5. Rozkład z budynku ASTOR według godziny i trasy	Średnia częstotliwość pojawiania się równa $\frac{1}{80}$
Godziny		$7, 8, 9, 10$	Możliwe do ustawienia godziny w środowisku (najciekawsze charakterystyki ruchu)
Współczynniki funkcji nagrody R	s_b	0.01	Kara za krok trwania aktywności przycisku, sumowana względem przycisków (z rozróżnieniem na góra i dół)
	s_d	12	Kara za nielegalny ruch
	s_f	12	Kara za dopuszczenie do zatrzymania środowiska
	pozostałe	0	

Tabela 7.13. Parametry środowiska z wymiarem czasu aktywności przycisków

Wprowadzony do opisanego środowiska agent wykorzystywał rozbudowany algorytm *Double Deep Q-Learning* z priorytetowym buforem pamięci. Jego parametry opisuje tabela 7.14. Wykresy zebranych w trakcie nauki metryk umieszczono na rysunku 7.8.

Parametr	Wartość	Opis
Liczba epizodów	1660	Długośc nauki, nieplanowanie przerwanej
Liczba kroków na epizod	20	
Współczynnik γ	0.99	Wartościuje nagrody natychmiastowe do przyszłych (ang. *discount rate*)
Początkowa wartość ε	1	
Końcowa wartość ε	0.1	Wartość ε co epizod nauki maleje liniowo o ustaloną wartość aż po 200 epokach osiągnie wartość minimalną
Krok zmniejszania ε	0.0045	
Minimalna wielkość bufora pamięci	50000	Liczba krotek doświadczenia, po których zebraniu rozpoczyna się proces nauki
Maksymalna wielkość bufora pamięci	1000000	Liczba ostatnich krotek przechowywanych w buforze, równa 50 ostatnim epizodom
Okres pobierania próbki z bufora mierzony liczbą kroków	4	Liczba kroków co które pobiera się próbkę z bufora
Wielkość próbki	32	Liczba losowych kotek pobieranych z bufora
Interwał kopiowania	40000	Liczba kroków, co które następuje kopiowanie wag pomiędzy sieciami w wariancie *Double Q-Learning*
Architektura sieci	*feedforward*	Sieć modelująca funkcję Q w wariancie *Deep Q-Learning*
	21	Długość wektora stanu
Wielkości kolejnych warstw sieci	128, 64, 16	Trzy warstwy ukryte
	3	Liczba możliwych akcji
Algorytm nauki sieci	*Adam*	Algorytm odpowiedzialny za aktualizację wag sieci, jest rozszerzeniem *SGD*
Funkcja błędu	*WMSE*	Funkcja minimalizowana przez sieć, ważony błąd średniokwadratowy (*Weigted MSE*)

Tabela 7.14. Parametry algorytmu textitDouble Deep *Q-Learning* z priorytetowym buforem pamięci i strategią ε-*greedy* wykorzystanego przez agenta w środowisku z wymiarem czasu aktywności przycisków



Rys. 7.8. Wykresy metryk zbieranych w trakcie nauki agenta w środowisku z wymiarem czasu aktywności przycisków. Wartości zarejestrowały zadowalające przebiegi. Jednak pomimo szybkiego osiągnięcia *plateau*, wartości nagrody, oraz średniego czasu oczekiwania nadal wykazują trend poprawy. Prawdopodobnie agent nie zakończył procesu korekcji strategii.

Źródło: opracowanie własne

8. Testy i interpretacja rezultatów

W celu oceny działania agentów przeprowadzono z ich udziałem testy i porównano osiągi z klasycznymi algorytmami kontroli windy. W tym rozdziale opisano sposób realizacji testów oraz przedstawiono i skomentowano ich rezultaty.

8.1. Realizacja testów

W trybie testowym wytrenowani agenci wprowadzani byli do tego samego środowiska, w którym przebywali w trakcie nauki. Wykorzystywali wówczas strategię π wyprowadzoną z nauczonych wartości reprezentacji funkcji Q:

$$\pi(s) = \arg\max_a Q(s, a)$$

Oznacza to, że w trakcie testów agent zawsze wybierze akcję, która według jego wiedzy skutkuje największą oczekiwaną skumulowaną przyszłą nagrodą. Rezygnuje z eksploracji na rzecz pełnej eksploatacji.

Po przeprowadzeniu symulacji testowej sterowanej zgodnie ze strategią wypracowaną przez agenta, uruchamiano ją ponownie z jednakowymi parametrami, ale sterowaną zgodnie z algorytmami heurystycznymi kontroli windy, opisanymi w sekcji 4.2:

1. algorytm najdłuższej kolejki *LQF*,
2. algorytm kolektywny *CA*,
3. algorytm ostatniej misji *LM*.

W trakcie trwania testowej symulacji zbierane były wartości metryk opisujących wyniki danej strategii, które porównywano wśród strategii operujących w jednakowym środowisku. Wspomnianymi metrykami były:

– średnia wartość nagrody przypadającej na jeden krok symulacji,

– średni czas oczekiwania obsłużonych pasażerów (czas przebywania w systemie),

– liczba obsłużonych pasażerów,

– liczba zatrzymań środowiska,

– liczba nielegalnych ruchów.

8.2. Rezultaty

Dla każdej opisanej w rozdziale 7 pary środowiska i agenta została przeprowadzona symulacja testowa dla strategii wypracowanej przez agenta. W tej sekcji przedstawiono ich rezultaty oraz porównano je z rezultatami osiągniętymi przez strategie heurystyczne w jednakowej symulacji.

8.2.1. Pojedynczy pasażer

W najprostszym środowisku, opisanym w sekcji 7.1 można bez trudu wyprowadzić optymalną strategię. Potrzebuje ona obsłużyć jedynego pasażera zwożąc go w najkrótszym czasie z 20 piętra na 1. Proces to spełniający trwa 40 kroków i przebiega następująco: wjazd windą na samą górę (19 kroków), zatrzymanie i wpuszczenie pasażera (1 krok), zjazd na sam dół (19 kroków), zatrzymanie i wypuszczenie pasażera prowadzące do zatrzymania środowiska i jego zrestartowania (1 krok). Opisany ciąg akcji skutkuje zsumowaną nagrodą o wartości $2 \cdot 19 \cdot (-1) + 100 = 62$, czyli średnia nagroda przypadająca na jeden krok jest równa $\frac{62}{40} = 1.55$. Sam pasażer przebywa w systemie pełnych 39 kroków.

W tabeli 8.1 widnieją wartości metryk zebranych w trakcie jednakowych symulacji opisanego środowiska przeprowadzonych dla strategii heurystycznych oraz strategii wypracowanej przez agenta, który uczył się w tym środowisku. Symulacja trwała 10000 kroków. Wszystkie algorytmy osiągają jednakowe i optymalne wyniki.

Strategia	**Obsłużeni pasażerowie**		**Średnia nagroda**	**Liczba zatrzymań środowiska**	**Liczba nielegalnych ruchów**
	Średni czas oczekiwania	**Liczba**			
LQF	39	250	1.55	250	0
CA	39	250	1.55	250	0
LM	39	250	1.55	250	0
Agent *QL*	39	250	1.55	250	0

Tabela 8.1. Wartości metryk w zależności od użytej strategii w trwającej 10,000 kroków testowej symulacji środowiska z pojedynczym pasażerem.

8.2.2. Nieostrożny wybór nagrody

Dla środowiska opisanego w sekcji 7.2 samodzielne zaprojektowanie strategii optymalnej jest już znacznie trudniejsze. Poprzez konstrukcję nagrody zależną od zmiany wartości sumy czasu aktywności przycisków zamierzano wymusić najczęstszą obsługę pasażerów. Okazała się ona jednak nieadekwatna do podstawowej metryki średniego czasu oczekiwania pasażerów, której minimalizacja jest celem nadrzędnym. Agent wypracował bowiem strategię zakładającą przetrzymywanie pasażerów i obsługiwanie ich możliwie najpóźniej. Skutkowała ona bardzo dużymi wartościami nagrody w momencie wypuszczenia pasażerów ze względu na dużą różnicę wartości sumy w następstwie takiego działania.

Tabela 8.2 zawiera porównanie wartości metryk w symulacji dla strategii agenta oraz strategii heurystycznych. Mimo obsłużenia dziesięciokrotnie mniej pasażerów oraz wartości średniego czasu ich oczekiwania większego o dwa rzędy wielkości, wartość średniej nagrody jest ponad stukrotnie większa. Pokazuje to jak ważne jest ostrożne wybranie nagrody w konstrukcji środowiska *RL*. Nagroda jest dla agenta, który nie jest bezpośrednio zaprogramowany do wykonywania zadania, jedyną informacją zwrotną o jakości podjętych decyzji. Potwierdza to również fakt, że agent nauczył się unikać akcji nielegalnych.

Strategia	**Obsłużeni pasażerowie**		**Średnia nagroda**	**Liczba nielegalnych ruchów**
	Średni czas oczekiwania	**Liczba**		
LQF	11.48	1203	0.802	0
CA	9.14	1282	0.653	0
LM	9.07	1278	0.637	0
Agent *QL*	572.05	155	8.023	0

Tabela 8.2. Wartości metryk w zależności od użytej strategii w trwającej 10,000 kroków testowej symulacji środowiska z nieostrożnie wybraną nagrodą.

8.2.3. Rozkład ASTOR według trasy

Środowisko opisane w sekcji 7.3 bazowało na rzeczywistym rozkładzie pojawiania się pasażerów. Było jednak wymagające ze względu na średni okres ukazania się użytkownika windy równy około 5 krokom.

W tabeli 8.3 przedstawiono rezultaty badanych strategii. Wypracowana przez agenta osiąga lepsze wyniki od algorytmów heurystycznych we wszystkich metrykach. W stosunku do algorytmu ostatniej misji uzyskuje o 1% mniejszy średni czas oczekiwania pasażerów przy jednoczesnym obsłużeniu większej liczby. Algorytm ostatniej misji wykorzystuje rozróżnienie między wezwaniami w górę i w dół, gdy decyduje o zatrzymaniu na piętrze pośrednim. Jednocześnie wektor reprezentacji stanu dostępny agentowi nie dostarcza tego rozróżnienia.

Strategia	**Obsłużeni pasażerowie**		**Średnia nagroda**	**Liczba nielegalnych ruchów**
	Średni czas oczekiwania	**Liczba**		
LQF	17.89	1342	-2.54	0
CA	12.54	1601	-2.05	0
LM	12.19	1577	-1.98	0
Agent *QL*	12.03	1603	-1.93	0

Tabela 8.3. Wartości metryk zebranych w zależności od wybranej strategii w trwającej 10,000 kroków testowej symulacji środowiska z rozkładem pojawiania się pasażerów z budynku ASTOR według obranej trasy.

8.2.4. Stan z rozróżnieniem przycisków w górę i w dół

W środowisku opisanym w sekcji 7.4 zmniejszono częstotliwość pojawiania się pasażerów oraz uwzględniono w reprezentacji jego stanu rozróżnienie względem przycisków góra/dół. Umieszczono w nim trzech agentów, różniących się wybranym wariantem *Experience Replay*. W jednakowej symulacji testowej porównano ich osiągi względem siebie oraz z algorytmami heurystycznymi (tabela 8.4).

Każdy z agentów wypracował strategię obsługującą więcej pasażerów i w krótszym średnim czasie niż algorytmy heurystyczne. Względem osiągającego najlepsze wyniki algorytmu ostatniej misji agenci zanotowali poprawę średniego czasu oczekiwania między 6.5% a 7.8%. Natomiast w porównaniu do algorytmu najdłuższej kolejki wartość ta sięga ponad 23%.

Agenci osiągają bardzo zbliżone do siebie wyniki, jednak w efekcie nauki o różnej długości i charakterze. Najdłuższego treningu wymagał algorytm wykorzystujący podstawowy bufor pamięci (10 tysięcy epok), krótszego używający pałac pamięci (8 tysięcy epok) a najkrótszego wariant przechowujący krotki doświadczenia w buforze priorytetowym (tysiąc epok).

8.2.5. Rozkład ASTOR według godziny i trasy

Środowisko opisane w sekcji 7.5 wykorzystuje rozkład zależny od godziny symulacji. Ruch w badanym budynku ASTOR jest widocznie zmienny w trakcie dnia. Dzięki dodatkowemu wymiarowi agent był w stanie wypracować osobną strategię dla każdej godziny.

Strategia			Obsłużeni pasażerowie		Średnia nagroda	Liczba nielegalnych ruchów
			Średni czas oczekiwania	Liczba		
LQF			12.16	897	-0.118	0
CA			10.16	936	-0.102	0
LM			10.07	929	-0.101	0
Agent *DDQL*	bufor pamięci	(7.4.1)	9.40	955	-0.095	0
	pałac pamięci	(7.4.2)	9.39	965	-0.096	0
	priorytetowy bufor pamięci	(7.4.3)	9.28	956	-0.094	0

Tabela 8.4. Wartości metryk w zależności od użytej strategii w trwającej 10,000 kroków testowej symulacji środowiska z rozróżnieniem przycisków góra/dół.

Osiągi agenta i ich porównanie do klasycznych algorytmów zawiera tabela 8.5. W trakcie trwającej 10,000 kroków testowej symulacji zarejestrował lepsze wartości wszystkich metryk poza liczbą obsłużonych pasażerów. Uzyskał również poprawę średniego czasu oczekiwania w wysokości 6% względem algorytmu ostatniej misji.

Strategia	Obsłużeni pasażerowie		Średnia nagroda	Liczba nielegalnych ruchów
	Średni czas oczekiwania	Liczba		
LQF	8.07	121	-0.364	0
CA	7.96	121	-0.363	0
LM	7.92	121	-0.363	0
Agent *DDQL*	7.42	120	-0.042	0

Tabela 8.5. Wartości metryk w zależności od użytej strategii w trwającej 10,000 kroków testowej symulacji środowiska z rozkładem pojawiania się pasażerów z budynku ASTOR według trasy i godziny. Godzina systemu zmienia się regularnie w trakcie symulacji.

8.2.6. Stan z wymiarem czasu aktywności przycisków

Najbardziej rozbudowane środowisko opisane w sekcji 7.6 rozwijało wektor stanów aktywności przycisków o czas ich trwania. Uzyskano tym samym znacznie więcej informacji jednak kosztem rozszerzenia przestrzeni możliwych stanów środowiska. W efekcie proces nauki staje się proporcjonalnie bardziej wymagający i dłużej zbiega do optimum.

W symulacjach testowych o długości 1000 kroków agent osiąga lepsze wartości metryk od algorytmów klasycznych. Ich porównanie pokazuje tabela 8.6. Poprawa średniego czasu oczekiwania względem algorytmów ostatniej misji i kolektywnego wynosi 2%, natomiast w porównaniu do algorytmu najdłuższej kolejki prawie 5%. Agent ten nie doprowadził również do zatrzymania środowiska poprzez nieobsłużenie pasażera w przedziale 50 kroków.

Strategia	Obsłużeni pasażerowie		Średnia nagroda	Liczba zatrzymań środowiska
	Średni czas oczekiwania	Liczba		
LQF	9.91	22	-0.014	0
CA	9.64	22	-0.013	0
LM	9.64	22	-0.013	0
Agent *DDQL*	9.43	21	-0.011	0

Tabela 8.6. Wartości metryk w zależności od użytej strategii w trwającej 1000 kroków testowej symulacji środowiska z wymiarem czasu aktywności przycisków. Godzina systemu zmienia się regularnie podczas przebiegu symulacji.

9. Podsumowanie

Działania podjęte w trakcie realizowania niniejszej pracy pozwoliły na osiągnięcie wszystkich jej założeń i celów. Wykorzystano szereg algorytmów uczenia ze wzmacnianiem w celu stworzenia i wytrenowania agentów zdolnych sterować windą w sposób osiągający widoczną poprawę względem istniejących algorytmów heurystycznych.

Zaimplementowano uniwersalny symulator systemu kontroli windy, który, wraz z zamodelowanymi rozkładami ruchu opracowanymi na podstawie danych z inteligentnego budynku ASTOR, pozwolił na skonstruowanie wielu środowisk o zróżnicowanym poziomie skomplikowania. Wykorzystane w procesie nauki do interakcji z agentami, dostarczały im potrzebnych informacji o stanie systemu, reagowały na ich wybory akcji i poprzez nagrodę wartościowały jakość decyzji. Utworzone w projekcie narzędzie pozwala na pełną kontrolę nad konstrukcją i parametrami środowiska, które są kluczowe w procesie szukania przez agenta optymalnej strategii względem nadrzędnego celu sterowania. Przeprowadzono wiele eksperymentów z wykorzystaniem różnorodnych algorytmów pochodnych do Q-*Learning*, umożliwiając ich porównanie i ocenę wpływu wykorzystania konkretnych parametrów algorytmu oraz zaawansowanych technik na naukę i osiągane rezultaty. Eksperymenty te zaowocowały wytrenowaniem agentów, których wypracowane strategie wykazują lepsze rezultaty w sterowaniu windą niż rozwiązania klasycznych metod kontroli.

9.1. Wnioski

Sterowanie systemem windy jest przykładem procesu decyzyjnego, który jest jednocześnie prosty koncepcyjnie i wymagający w optymalizacji. Przeprowadzone w pracy badania wykazały przydatność algorytmów uczenia ze wzmocnieniem w wypracowaniu strategii maksymalizującej pewną zwrotną wartość w takim procesie. Zauważono jak ważne dla osiągnięcia nadrzędnej optymalizacji jest odpowiednie i ostrożne skonstruowanie funkcji nagrody, która jest jedyną informacją o dokładności decyzji agenta. Agent maksymalizuje bowiem skumulowaną wartość nagrody, może się więc okazać, że nie jest to dokładnie zbieżne z założonym celem wyjściowego systemu.

Proces decyzyjny optymalizowany przez agenta może być zarówno bardzo prosty posiadający kilkadziesiąt stanów, jak i rozbudowany, którego przestrzeń stanów ma liczbę unikalnych reprezentacji rzędu 10^{35}. Im jednak bardziej rozbudowane środowisko tym proces nauki jest dłuższy. W algorytmach pochodnych *Q-Learning* może być skrócony lub ustabilizowany poprzez wykorzystanie metod takich jak *Double Q-Learning*, *Experience Replay* lub sieci neuronowych. Sieci neuronowe są przydatne zwłaszcza do interpolacji funkcji wartości akcji w środowisku z dużą liczbą stanów. W procesie nauki wymagana jest również zrównoważona eksploracja środowiska, która może być zapewniona przez używanie strategii *ε-greedy*.

Przeprowadzone w jednakowym środowisku eksperymenty nauki agentów przy wykorzystaniu algorytmu *Double Deep Q-Learning* różniącego się strukturami pamięci pozwoliły na wyizolowanie wpływu technik *Experience Replay* na proces nauki. Najszybszy i najbardziej stabilny proces nauki zaobserwowano przy wykorzystaniu priorytetowego bufora pamięci, nieco wolniejsze, lecz nadal stabilne, okazało się użycie pałacu pamięci, a najwolniejsze i najmniej stabilne zastosowanie zwykłego bufora pamięci.

Lepsze wyniki opracowanych przez agentów strategii zasadzały się w możliwości niebezpośredniego wykorzystania rozkładu prawdopodobieństwa pojawiania się pasażerów. Niebezpośredniego, ponieważ strategia jest efektem wielokrotnych, drobnych zmian funkcji nagrody otrzymywanej w środowisku o dynamice zależnej od wspomnianego rozkładu. W przeciwieństwie do algorytmów heurystycznych, agent nie jest odgórnie zaprogramowany do wykonywania zadania ale bardziej programuje się w trakcie nauki.

9.2. Możliwości rozwoju projektu

Dalszy rozwój projektu może zakładać implementację innych algorytmów uczenia ze wzmacnianiem, również z poza grupy *Value-based*, i zbadania ich własności. Stworzenie również środowiska bazującego na rozkładzie zależnym od większej liczby czynników takich jak dzień tygodnia może dalej poprawić osiągi agentów.

Problem sterowania windą może być łatwo przekonwertowany na szereg problemów transportowych, na przykład optymalnego odbierania i składowania produktów z kilku linii produkcyjnych. Wiedza zdobyta w tej pracy pozwoli wytrenować agenta zdolnego wypracować strategie obsługi takich systemów.

Utworzone w pracy narzędzie jest dobrym punktem wyjścia do badania procesów decyzyjnych. Zaimplementowane algorytmy uczenia ze wzmacnianiem mogą być wygodnie wykorzystane do trenowania agentów również w zupełnie innych zadaniach, dzięki implementacji środowiska zgodnie z zaproponowanymi przez *OpenAI Gym* wytycznymi interfejsu środowiska *RL* [13].

Bibliografia

[1] Robert Crites and Andrew Barto. “Improving Elevator Performance Using Reinforcement Learning”. In: *Advances in Neural Information Processing Systems*. Ed. by D. Touretzky, M. C. Mozer, and M. Hasselmo. Vol. 8. MIT Press, 1996.

[2] Robert H. Crites and Andrew G. Barto. “Elevator Group Control Using Multiple Reinforcement Learning Agents”. In: *Machine Learning* 33.2 (Nov. 1998), pp. 235–262. ISSN: 1573-0565. DOI: *10.1023/A:1007518724497*.

[3] Li Deng and Dong Yu. *Deep Learning: Methods and Applications*. Tech. rep. MSR-TR-2014-21. Microsoft, May 2014.

[4] Hado Hasselt. “Double Q-learning”. In: *Advances in Neural Information Processing Systems*. Ed. by J. Lafferty et al. Vol. 23. Curran Associates, Inc., 2010.

[5] Hado van Hasselt, Arthur Guez, and David Silver. *Deep Reinforcement Learning with Double Q-learning*. 2015. arXiv: *1509.06461* [cs.LG].

[6] Dan Horgan et al. *Distributed Prioritized Experience Replay*. 2018. arXiv: *1803.00933* [cs.LG].

[7] IBM. *The Smarter Buildings Survey*. 2010.

[8] N. Imasaki et al. “Elevator group control system tuned by a fuzzy neural network applied method”. In: *Proceedings of 1995 IEEE International Conference on Fuzzy Systems.* Vol. 4. 1995, 1735–1740 vol.4. DOI: *10.1109/FUZZY.1995.409916*.

[9] Vijay R Konda and John N Tsitsiklis. “Actor-critic algorithms”. In: *Advances in neural information processing systems*. 2000, pp. 1008–1014.

[10] Daeyeol Lee, Hyojung Seo, and Min Whan Jung. “Neural Basis of Reinforcement Learning and Decision Making”. In: *Annual Review of Neuroscience* 35.1 (2012), pp. 287–308. DOI: *10.1146/annurev-neuro-062111-150512*.

[11] Sandor Markon, Hajime Kita, and Yoshikazu Nishikawa. “Adaptive Optimal Elevator Group Control by Use of Neural Networks”. In: 1994.

[12] Volodymyr Mnih et al. “Human-level control through deep reinforcement learning”. In: *Nature* 518 (2015), pp. 529–533.

[13] OPENAI. *Gym – a toolkit for developing and comparing reinforcement learning algorithms. https://gym.openai.com/docs/*. [Online].

[14] Bernhard Seckinger and Jana Koehler. “Online synthesis of elevator controls as a planningproblem.” In: *Thirteenth Workshop on Planning and Configuration.* Department of Computer Science, University of Wuerzburg, Jan. 1999.

[15] Marja-Liisa Siikonen. “Planning and Control Models for Elevators in High-Rise Buildings”. PhD thesis. Jan. 1997.

[16] Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. Second. The MIT Press, 2018.

[17] Michel Tokic and Günther Palm. “Value-Difference Based Exploration: Adaptive Control between Epsilon-Greedy and Softmax”. In: *KI 2011: Advances in Artificial Intelligence*. Springer Berlin Heidelberg, 2011, pp. 335–346. DOI: *10.1007/978-3-642-24455-1_33*.

[18] Oriol Vinyals et al. “Grandmaster level in StarCraft II using multi-agent reinforcement learning”. In: *Nature* (2019), pp. 1–5. DOI: *10.1038/s41586-019-1724-z*.

[19] Hua Wei et al. “IntelliLight: A Reinforcement Learning Approach for Intelligent Traffic Light Control”. In: *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. KDD ’18. London, United Kingdom: Association for Computing Machinery, 2018, pp. 2496–2505. ISBN: 9781450355520. DOI: *10.1145/3219819.3220096*.

[20] Wikipedia contributors. *Artificial intelligence — Wikipedia, The Free Encyclopedia*. *https://en.wikipedia.org/w/index.php?title=Artificial_intelligence&oldid=1064278900*. [Online]. 2021.