AKADEMIA GÓRNICZO-HUTNICZA IM. STANISŁAWA STASZICA W KRAKOWIE
WYDZIAŁ ELEKTROTECHNIKI, AUTOMATYKI,
INFORMATYKI I INŻYNIERII BIOMEDYCZNEJ
KATEDRA AUTOMATYKI I INŻYNIERII BIOMEDYCZNEJ

Praca dyplomowa magisterska

System inteligentnego kontekstowego wnioskowania i sterowania grą komputerową, wykorzystujący asocjacyjny grafowy model danych i metody inteligencji obliczeniowej.

Intelligent system of contextual inference and controlling of a computer game using an associative graph data model and methods of computational intelligence.

Autor: *Agnieszka Konsur*
Kierunek studiów: *Automatyka i Robotyka*
Opiekun pracy: *dr hab. Adrian Horzyk*

Kraków, 2017

Uprzedzony o odpowiedzialności karnej na podstawie art. 115 ust. 1 i 2 ustawy z dnia 4 lutego 1994 r. o prawie autorskim i prawach pokrewnych (t.j. Dz.U. z 2006 r. Nr 90, poz. 631 z późn. zm.): „ Kto przywłaszcza sobie autorstwo albo wprowadza w błąd co do autorstwa całości lub części cudzego utworu albo artystycznego wykonania, podlega grzywnie, karze ograniczenia wolności albo pozbawienia wolności do lat 3. Tej samej karze podlega, kto rozpowszechnia bez podania nazwiska lub pseudonimu twórcy cudzy utwór w wersji oryginalnej albo w postaci opracowania, artystyczne wykonanie albo publicznie zniekształca taki utwór, artystyczne wykonanie, fonogram, wideogram lub nadanie.", a także uprzedzony o odpowiedzialności dyscyplinarnej na podstawie art. 211 ust. 1 ustawy z dnia 27 lipca 2005 r. Prawo o szkolnictwie wyższym (t.j. Dz. U. z 2012 r. poz. 572, z późn. zm.) „Za naruszenie przepisów obowiązujących w uczelni oraz za czyny uchybiające godności studenta student ponosi odpowiedzialność dyscyplinarną przed komisją dyscyplinarną albo przed sądem koleżeńskim samorządu studenckiego, zwanym dalej „sądem koleżeńskim", oświadczam, że niniejszą pracę dyplomową wykonałem(-am) osobiście i samodzielnie i że nie korzystałem(-am) ze źródeł innych niż wymienione w pracy.

<podpis dyplomanta>

Składam serdecznie podziękowania
Panu doktorowi Adrianowi Horzykowi
za życzliwość, zrozumienie oraz cenne wskazówki,
które pomogły mi w napisaniu pracy.

Spis treści

1.Wstęp

Rozrywka i rozwój są nieodłącznym elementem ewolucji. Sama potrzeba rozrywki towarzyszy człowiekowi od zarania dziejów. Nasi przodkowie, siedząc wieczorami przy ognisku, opowiadali historie o odległych i ekscytujących wydarzeniach. Wtedy to, za pomocą wyobraźni kierowano umysłami, przekazywano im obrazy i dźwięki. To te spotkania były największą atrakcją dnia. W pozostałej jego części trudzono się by przetrwać. Dzień człowieka współczesnego rządzi się tymi samymi prawami, co kiedyś. Większość dnia każdy z nas spędza w pracy, następnie musi wypełnić obowiązki domowe i dopiero wieczorem możemy znaleźć chwilę, na odpoczynek i relaks. Dzisiaj w celu utrzymania się, korzystamy ze zgoła innych cech. Dawniej zręczność wykorzystywana była w codziennej walce o życie, a chęci walki dawano upust w rutynowych polowaniach. Te instynkty towarzyszą nam i obecnie, ale ze względu na rozwój zostały one stłumione. Natomiast, szeroko pojęta rozrywka wykorzystuje je w pełni. Dzisiejsze programy telewizyjne i gry komputerowe wypełnione są opowieściami o legendarnych krainach, wojnach ludów czy mitycznych stworzeniach walczących o przeżycie. Człowiek w wirtualnym świecie ma możliwość doświadczać, czego tylko zapragnie, a stworzona rzeczywistość im bardziej jest pełna kontrastów tym atrakcyjniejsza. Poza oczywistymi efektami, grafiką i animacjami, które wraz z rozwojem komputerów udoskonalane były od lat, obecnie położono też duży nacisk na logikę gier. To skomplikowane algorytmy są teraz w dużej mierze odpowiedzialne za doświadczenia gracza. A co może być ciekawsze niż walka z innym zawodnikiem? Oprócz znacznego rozwoju gier typu „multiplayer" rozpoczęto również wprowadzać sztuczną inteligencję do logiki gry.

1.1. Cel i zakres pracy

W skład mojej pracy magisterskiej wchodzą dwie wzajemnie się uzupełniające całości. Pierwszą z nich jest gra komputerowa z zaimplementowanym asocjacyjnym modelem danych, drugą natomiast stanowi niniejsza praca, która jest kompletnym opisem programu, jego projektu oraz wszystkich użytych technik.

Celem części praktyczniej jest stworzenie stosunkowo prostej gry komputerowej typu „Snake", wyposażonej w moduł sztucznej inteligencji. W uogólnieniu inni „automatyczni" zawodnicy gry mają poruszać się kierowani przez utworzony system skojarzeniowy. Logika gry oparta będzie o asocjacyjny grafowy model danych. W celu uatrakcyjnienia rozgrywki, system będzie posiadał zdolność do uogólniania, a następnie formowania wiedzy o następnym ruchu. Wykorzystywał będzie również inteligentne kontekstowe wnioskowanie. Sama aplikacja docelowo ma zawierać prostą grafikę, przyjemną dla oka gracza.

Dokument ten ma na celu przedstawienie wykonanych prac, zarówno od strony teoretycznej jak i praktycznej. Pierwsze dwa rozdziały opisują tematykę projektu i wgłębiają się w historię gier komputerowych. Następnie dużą część mojej pracy poświęciłam na opis głównych algorytmów oraz możliwości ich zastosowania. Rozdział trzeci definiuje pojęcie asocjacji oraz związane z nim mechanizmy skojarzeniowe. Następnie w rozdziale czwartym opisałam wybrane przeze mnie środowisko pracy i język programowania oraz uzasadniłam moją decyzję. Rozdziały piąty, szósty i siódmy są bezpośrednim opisem zaimplementowanej gry. Przedstawiona została jej logika oraz architektura jak i realizacja opisanych wcześniej koncepcji. Pracę zakończyłam krótkim podsumowaniem, w którym wymieniłam najważniejsze kwestie poruszane we wcześniejszych rozdziałach i na ich podstawie przedstawiłam wnioski końcowe, dotyczące implementacji gier komputerowych z wykorzystaniem silników sztucznej inteligencji.

1.2. Założenia i spodziewane wyniki

Wynikiem niniejszej pracy była implementacja gry komputerowej oraz systemu inteligentnego kontekstowego wnioskowania i sterowania nią. Logika gry została oparta na asocjacyjnym grafowym modelu danych, który umożliwił implementację mechanizmów skojarzeniowych. Klasa zawodnika została wyposażona w algorytmy umożliwiające szybkie wnioskowanie na temat otaczających ją obiektów oraz pozycji gracza. Założono, iż „automatyczni i inteligentni" współgracze będą sprawnie rozpoznawać pożywienie i szybko reagować na zmiany położenia rywali czy statycznych ograniczeń pola gry, np. muru. Funkcje te zostały wprowadzone również za pomocą zdolności do analizy kontekstowej i uogólniania. Wprowadzone sterowanie odbywa się w sposób kreatywny i atrakcyjny dla gracza z zaimplementowane algorytmy działają wydajnie. Podejmowane decyzje dotyczące kierunku jak i sam ruch, są płynne.

2. Tematyka projektu

2.1. Gra komputerowa

Gra komputerowa według słownika języka polskiego [1] jest to „*Gra rozgrywana na ekranie komputera; też: program komputerowy umożlwiający tę grę*”. Definicja sama w sobie jest prosta, co za tym idzie, nie określa sposobu klasyfikacji gier. Gry możemy dzielić ze względu na platformę docelową ich użytkowania, czy też segregować je z perspektywy programowania. Jeśli na myśli mamy grę w szerszym znaczeniu, w którą grać można na komputerze, konsoli, telefonie itd., powinniśmy częściej używać sformułowania „gra wideo” niż popularnego zwrotu „gra komputerowa”, która zgodnie z pierwszą definicją uruchamiana może być tylko na ekranie komputera, a więc komputerze stacjonarnym czy laptopie. Jednakże biorąc pod uwagę, że w dzisiejszych czasach możliwości migracyjne oprogramowania między różnymi platformami są bardzo duże i stają się normą, granica między grami komputerowymi a grami wideo zdaje się zacierać. W niniejszej pracy jak i też osobiście uważam grę komputerową za formę aplikacji, więc oba te terminy będę stosować w sposób wymienny.

Czym jest więc gra komputerowa? Według zbadanych przeze mnie źródeł, jest to program do celów rozrywkowych lub edukacyjnych, który wymaga od użytkownika rozwiązywania zadań logicznych lub zręcznościowych. W praktyce gra komputerowa wykorzystuje wszelkie możliwości, jakie oferuje współczesna technika, tak aby kontakt z nią uczynić możliwie ciekawym, interaktywnym i pożytecznym. Gry możemy podzielić w zależności od platformy sprzętowej, jak i ze względu na zadania stawiane przed graczem. Zadania te różnią się w zależności od gatunku, mogą polegać na rozwiązaniu zadania logicznego, tworzenia strategii, eliminacji wirtualnych przeciwników, rywalizacji ze sztuczną inteligencją czy innymi graczami.

Jedną z głównych cech gry jest posiadany cel oraz towarzyszące mu wyzwania. Gra różni się od medium telewizyjnego swoją interaktywnością, która uatrakcyjniona jest formą audiowizualną. Gracz poprzez osobiste zaangażowanie ma wpływ na przebieg wydarzeń na ekranie. To on za pomocą klawiatury, myszki, dżojstika ekranu dotykowego czy innych specjalnych urządzeń odpowiada za działania jednostki organizacyjnej. Gry komputerowe wymagają również odbiornika wyświetlającego sygnał wideo, którym może być monitor, wyświetlacz, projektor czy telewizor.

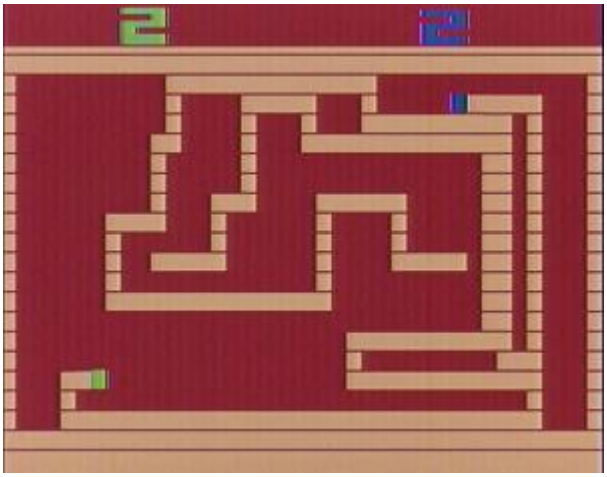
Gry wideo wywodzą się z gier planszowych i fabularnych oraz różnych dyscyplin sportowych. Prototyp pierwszej gry elektronicznej powstał w 1947 roku. Od tego czasu gry komputerowe były produktem akademickim, a do masowego obiegu weszły za sprawą gry „*Pong*” w roku 1972. Początkowo rynek gier wideo oparty był na automatach i konsolach, dopiero w latach osiemdziesiątych XX wieku rozszerzył się na komputery osobiste. Tam jednak gry elektroniczne nie cieszyły się taką popularnością, ponieważ były ciężko dostępne, duże

i drogie. Mimo ograniczeń technicznych rozpowszechniony został gatunek gier tekstowych, gdzie komendy gracza zapisywane były za pomocą tekstu. Prawdziwym przełomem technologicznym stało się wprowadzenie w 1982 roku komputera ZX Spectrum, który umożliwiał odtwarzanie dźwięku i posiadał kolorowy wyświetlacz. Dało to początek grom platformowym. Niewiele później w roku 1986 gra „*Super Mario Bros*" przyczyniła się do ich wielkiej popularności. Do roku 2000 rynek gier komputerowych rozwijał się bardzo intensywnie, powstały nowe gatunki gier, wykorzystujące najnowsze technologie. Pierwsza grafika trójwymiarowa pojawiła się w 1980 roku, a za sprawą gry „*Tomb Raider*" została bardzo rozpowszechniona. W roku 1994 zapoczątkowana została seria „*FIFA*". Upowszechnienie Internetu umożliwiło rozwój gier komputerowych wieloosobowych. Powstały takie hity jak „*Warcraft II*", „*Tides of Darkness*" czy „*Diablo*". Szósta generacja gier wideo rozpoczęła się z wprowadzeniem karty pamięci, która umożliwiła zapisanie stanu rozgrywki. W roku 2000 utworzona została popularna gra symulująca ludzkie życie „*The Sims*". W 2001 roku powstał Xbox Live, w 2005 Xbox 360 i PlayStation 3. Na przełomie wieków zaczął kształtować się bardzo dochodowy biznes gier komputerowych, który w dalszym ciągu się rozwija. Olbrzymią popularność zyskują gry typu „*Stream*" oraz przeglądarkowe. Rynek gier komputerowych poszerzył się o takie urządzenia jak telefony komórkowe oraz smartphony [2].

2.2. Gry typu „Snake"

Gra typu „Snake" występuje w dwóch głównych wariantach. Pierwszy charakteryzuje się długim podobnym do węża stworzeniem, które porusza się w celu zdobycia pożywienia. Nie może ono uderzyć głową o ściany otaczające plansze gry jak również części swojego ciała. Gdy zje kawałek „jedzenia" ciało stworzenia wydłuża się, co dodaje grze trudności. Gracz nie może „zatrzymać węża", jest on cały czas w ruchu, poruszany najczęściej za pomocą strzałek klawiatury. Drugi wariant gry zakłada istnienie dwóch lub większej ilości graczy, którzy poruszają się w planszy wypełnionej rywalami. Celem każdego gracza jest blokada pozostałych, tak by nie utracić przy tym życia. Gdy gracz jest jeden, dodakowe węże poruszają się za pomocą sztucznej inteligencji.

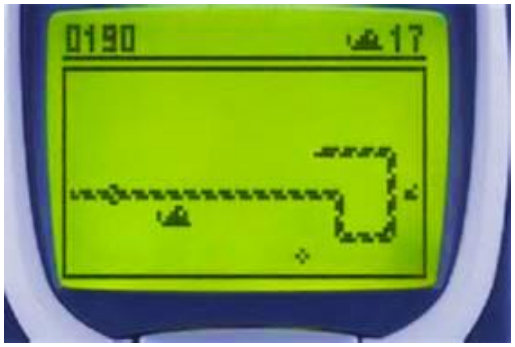
Pierwsza gra typu „*Snake*" powstała w roku 1976, nazywała się „*Blockade*" i wyprodukowana była przez firmę Gremlin Industries na automaty do gier. Kolejna jej wersja, premierowa dla komputerów osobistych nosiła nazwę „Worm" i była dokładną kopią wersji automatowej. Dopiero w roku 1982 do gry dołożono zwiększanie prędkości i utratę życia po jednym skuciu. Pierwsza wersja wieloosobowego Snake powstała w roku 1977 o nazwie „*Surround*" dla konsoli Atai 2600 [3]. Od tego czasu powstało wiele różnych odmian tej gry. Jedną z ostatnich jakie się pojawiły, jest jej nowoczesna odsłona, która została udostępniona w roku 2016 pod postacią przeglądarkowej gry „*Slither.io*".



Rys.1. Ekran gry „Surround" z 1977 roku

Rys.2. Ekran gry „Slither.io" z 2016 roku

Gry typu „Snake" podbiły nie tylko rynek komputerów osobistych. Telefony firmy „*Nokia*" wspominane jako wzór niezawodności, odporności i prostoty, wyposażone były również w tę kultową grę. W 1997 roku wraz z modelem „*Nokia 6110*" na rynku pojawiła się pierwsza wersja „*Snake*", wyposażona jedynie w czarne kwadraciki imitujące postać węża oraz możliwości ruchu w 4 kierunkach. W 2002 roku gra wyszła na kolorowym wyświetlaczu i za pomocą technologii Bluetooth umożliwiała rywalizację w trybie „*multiplayer*". Kolejne wersje rozbudowywane były o nowe wzory, efekty graficzne, a nawet grafikę 3D.



Rys.3. Ekran gry „Snake II" Nokia 3310

Rys.4. Ekran gry „Snake III" na telefonie Nokia w roku 2005

2.3. Tworzenie gier komputerowych

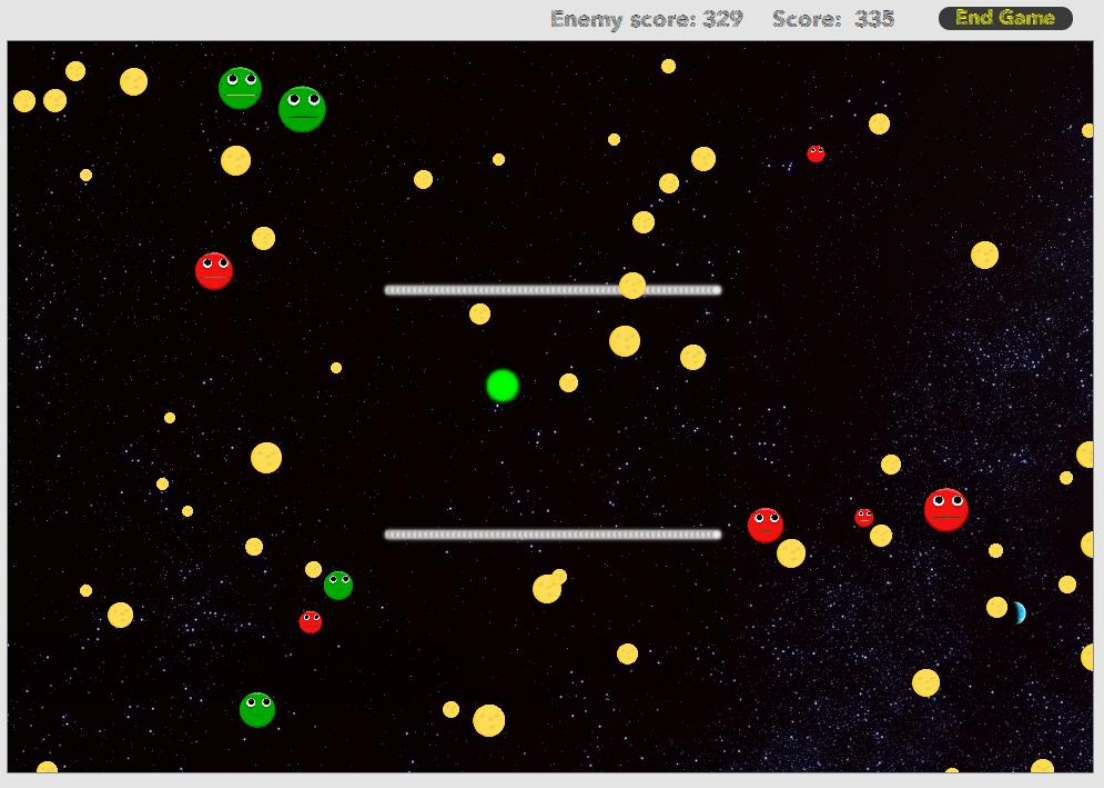
Jak już wcześniej wspominałam, że rynek gier komputerowych rozwija się bardzo dynamicznie, a skala przedsięwzięcia jakim jest stworzenie gry komputerowej diametralnie się powiększa. Gry z początku lat 90. XX wieku tworzone były przez pojedynczych programistów, później powstały zespoły developerskie, jednak były one niewielkie, ograniczały się do kilku osób. Dopiero rozwój techniczny komputerów zwiększył znacząco zapotrzebowanie na rozrywkę elektroniczną. Wraz z popytem, zespoły specjalizowały się i powiększały.

Obecnie liczebność zespołu pracującego nad grą, zależy od klasy powstającej produkcji. Standardowy czas tworzenia gry opartej na gotowym silniku, szacowany jest na kilka miesięcy, a do pracy angażowani są projektanci, graficy i programiści. W przypadku dużych gier, takich jak „*GTA 4*" czy „*Wiedźmin*", zespół może liczyć od kilkudziesięciu do stu kilkudziesięciu osób, a prace nad grą mogą trwać latami. Współcześnie za tworzenie gier, oprócz działu programistów, możemy wyróżnić działy projektantów, grafików i specjalistów od efektów

dźwiękowych oraz kontroli jakości. Pierwszy zespół odpowiedzialny jest za projekt. Zaprojektować należy fabułę gry, jej poziomy i mechanikę. Ważne jest również zbalansowanie rozrywki. Dział graficzny składa się z artystów, grafików, animatorów, dźwiękowców i techników, odpowiedzialnych za przygotowanie elementów gry od strony technicznej i graficznej. Ponieważ w produkcji gier nie jesteśmy w stanie generować testów automatycznych, dział testów jest niezbędny i szczególnie istotny w końcowej fazie projektu.

2.4. Projekt CleverWorm

„*Clever Worm*" jest moją interpretacją niedawno powstałych gier typu „*Snake*", takich jak „*Slither.io*" czy „*Agar.io*". W projekcie skupiłam się na wersji stacjonarnej, nieobejmującej trybu „*multiplayer*". Moim celem była implementacja algorytmów, pozwalających na bardziej atrakcyjne ruchy automatycznych współgraczy. Założeniem jest, że za pomocą grafów AGDS oraz analizy kontekstowej granie przeciwko A.I. stanie się bardziej wyzywające. Docelowo gra ma pobudzić użytkowników do myślenia i szybkich reakcji. Główną czynnością przewidzianą dla użytkownika jest wyszukiwanie pożywienia, a następnie zdobycie go przed innymi współgraczami. Za dodatkowy element rywalizacji można uznać, możliwość pozbycia się mniejszego rywala.



Rys.5. Ekran gry „Clever Worm"

3. Sztuczne systemy skojarzeniowe AAS

3.1. Sztuczna inteligencja

Narodziny komputera umożliwiły powstanie sztucznej inteligencji. Kluczową postacią we wczesnych etapach jej rozwoju był angielski matematyk, Alan Turing, który przyczynił się również do rozwoju komputera ogólnego użytku. Zdefiniował on abstrakcyjną maszynę składającą się z trzech głównych elementów - skrzynki odpowiedzialnej za obliczenia, taśmy papierowej oraz urządzenia, które zapisuje dane na taśmie. Udowodnił on, że jeśli ustalony zostanie odpowiedni algorytm i zapewniona zostanie nieskończenie długa taśma, to komputer będzie mógł wykonywać dowolnie zdefiniowany zbiór operacji. Każda maszyna tak określona, to uniwersalna maszyna Turinga. Powstałe wnioski stały się niepowtarzalne i dały początek rewolucji komputerowej. Ten sam człowiek uznał, że komputery mogą być inteligentne, więc aby to określić wprowadził znany test inteligencji maszyn, (tzw. test Turinga [4]), który mówił, że gdy maszyna potrafi zmylić rozmówcę w taki sposób, aby ten myślał, że rozmawia z drugim człowiekiem, wtedy z definicji komputer musi być inteligentny. Testy nad inteligencją maszyn stały się początkiem badań sztucznej inteligencji. Sam termin „*sztuczna inteligencja*" został zdefiniowany przez Johna McCarthy'ego w 1955 roku [4], jako „*konstruowanie maszyn, o których działaniu dałoby się powiedzieć, że są podobne do ludzkich przejawów inteligencji*". Naukowcy zauważyli, że „*najbardziej spektakularne cechy ludzkiej inteligencji wiążą się z manipulacjami na abstrakcyjnych symbolach*" a to właśnie potrafią robić komputery. Postanowiono opisać komórki nerwowe w kategoriach funkcji cyfrowych, które pozwalają wykonywać proste operacje logiczne: TAK, NIE, LUB. Następnie przez lata powstało wiele programów, które uchodziły za inteligentne, lecz w rzeczywistości wyspecjalizowane były w jednym zadaniu. Tworzono dowody matematyczne, systemy eksperckie, czy programy wygrywające z mistrzami szachów, a wiara w sztuczną inteligencję powoli zanikała. Eksperyment „*chińskiego pokoju*" [4] pokazał, że system nie musi być inteligentny, by tak się zachowywać. Obecnie uważa się, że komputer, mógłby przynajmniej teoretycznie symulować cały mózg [4]. Możliwe byłoby przecież zaimplementowanie wszystkich neuronów i połączeń między nimi, tak, aby nie istniała możliwość odróżnienia „inteligencji" mózgu od inteligencji „sztucznej" [4]. Według ankietowanych ekspertów, osiągnięcie AI na ludzkim poziomie nastąpi przed rokiem 2040 w 50%, a przed rokiem 2075 w 90% [5].

Współcześnie z pojęciem sztucznej inteligencji możemy wiązać wiele różnych metod i technik obliczeniowych, gdzie nie wszystkie łączą się z inteligencją ludzką. Ponieważ programy te nie są zdolne automatycznie przetwarzać informacje i formułować wnioski, jak odbywa się to w ludzkim mózgu, więc wyodrębniono je pod postacią tzw. inteligencji obliczeniowej. Inteligencję obliczeniową możemy podzielić ze względu na dwa podejścia do

pracy nad nią. Pierwsze z nich jest podejściem symbolicznym, które zakłada tworzenie modeli matematyczno-logicznych analizowanych problemów, a następnie przedstawianie ich w postaci programów komputerowych, mających realizować określone funkcje, które mogą być uważane za składowe inteligencji. W tej grupie znajdują się takie metody jak algorytmy generyczne czy metody logiki rozmytej. Druga grupa reprezentuje podejście subsymboliczne, polegające na tworzeniu programów i struktur „samouczących się", opracowywanie procedur „uczenia maszynowego". Bazują one na sieciach neuronowych i asocjacyjnych [6].

Aktualnie metody inteligencji obliczeniowej znajdują praktyczne zastosowania w uczeniu maszynowym, eksploracji danych, rozpoznawaniu obrazów, rozpoznawaniu mowy, maszynowym tłumaczeniu tekstów, aproksymacji, prognozowaniu i regresji.

3.2. Dane i informacje

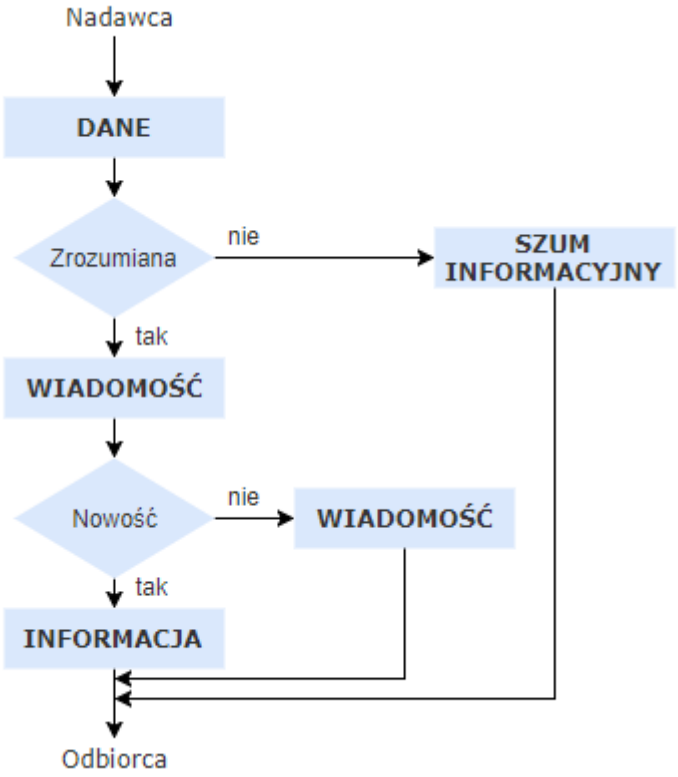
Dane to pewne zbiory wartości (np. liczbowe, logiczne, znakowe), które są ze sobą w jakiś sposób powiązane. Dane mogą być gromadzone i przechowywane w różnych strukturach np. tabelach, stosach, stringach, kolejkach, listach, drzewach czy grafach. Każdą wartość możemy w informatyce przedstawić za pomocą prostych typów danych, które występują we wszystkich językach programowania (tj. boolean, integer, real, char). Dane przesyłane są do świadomości odbiorcy w postaci komunikatu. Każdy komunikat zawiera dane, ale dane same w sobie nie mają znaczenia ani celu. Dane odwzorowują różne wielkości fizyczne lub abstrakcyjne i mogą być powiązane różnymi relacjami. Jeśli dane odebrane przez system sensoryczny są ze sobą w jakiś sposób powiązane, a odbiorca zna znaczenie tego powiązania, te dane tworzą dla niego pewną informację. Jeśli dane nie powodują zmiany stanu, wiedzy czy sposobu działania, pozostają pewną kombinacją, która dla odbiorcy nie ma wartości informacyjnej. Przez układ danych, można rozumieć rozciągniętą w czasie kombinację danych, w której poszczególne dane, docierają do systemów sensorycznych, co pewien określony odstęp czasu [7].

Informacja jest najczęściej określana, jako zbiór powiązanych ze sobą danych, któremu można przypisać sens lub znaczenie. Ważnym jej elementem jest zmniejszenie niewiedzy odbiorcy czy entropii układu. Znaczenie informacji jest wynikiem interpretacji odbiorcy. Równanie Langeforsa prezentuje jak na informację (I) wpływa:

$$I = i(D, S, t) \qquad (3.2.1)$$

i – proces interpretacji, D- dane, S- przedwiedza, t- czas.

Informacja powinna wnosić do świadomości element nowości. Chcąc by dane stały się wiedzą, potrzebny jest odbiorca, który decyduje jak dane zinterpretować, czy są one zrozumiałe oraz w jakim stopniu. Wtedy dane stają się wiadomością. Następnie odbiorca powinien określić, czy wiadomość jest powtórzeniem czegoś, co już wie, jak również czy stanowi dla niego element nowości [8].



Rys.6. Dane, wiadomości i informacje

Informacja charakteryzuje się następującymi cechami:

- celowość – informacja musi czemuś lub komuś służyć;
- rzetelność – dotyczy prawdziwości źródła jak i jego zawartości;
- aktualność – informacja musi dotyczyć okresu decyzyjnego i być dostarczona w odpowiednim czasie;
- kompletność – informacja musi uwzględniać kontekst decyzyjny, nie może być wyrywkowa;
- wszechstronność – powinna przedstawiać sytuację decyzyjną z różnych punktów widzenia;
- granulowość – informacja nie powinna być zbyt szczegółowa, ani zbyt ogólna;
- uzasadnione pokłady finansowe – wykorzystanie informacji, powinno przynosić korzyści przynajmniej pokrywające nakłady poniesione na jej zdobycie[8].

Każda informacja powinna wpływać na odbiorcę poprzez zmianę jego stanu, zmianę wiedzy czy sposobu działania. Powinna powodować reakcję wewnętrzną lub zewnętrzną. Jak już zostało wspominane, reakcja ta zależy od interpretacji odbiorcy, więc dla jednego człowieka zbiór danych będzie informacją a dla innego nie, ze względu na inny kontekst posiadanej wiedzy. Z punktu widzenia systemów komputerowych układ danych może zostać uznany za informację, kiedy docierające do systemu dane zostaną odebrane i zrozumiane oraz nastąpi zwykle wcześniej zdefiniowana reakcja, która wpłynie na zmianę stanu lub działania systemu.

Wynika z tego, że wiedza i inteligencja działania większości programów informatycznych zakodowana jest w ich warunkach i algorytmach, które umożliwiają im reagowanie na docierające do nich dane. Większość takich reakcji jest statyczna, deterministyczna i z góry przewidziana, a układy same w sobie nie posiadają zdolności samoadaptacji. Powoduje to, że system informatyczny może odbierać i przekazywać tylko określone przez algorytm informacje. Wiedza systemu kształtowana jest na podstawie docierających informacji oraz ich weryfikacji. Proces jej kształtowania można określić z pomocą rysunku numer 7 [8].

Rys.7. Kształtowanie się wiedzy

3.3. Asocjacja

3.3.1. Mechanizmy skojarzeniowe

Wiemy, że wywołanie skojarzenia lub sekwencji skojarzeń jest spowodowane bezpośrednim dostępem do danych oraz podaniem odpowiedniego kontekstu dla umysłu. Ciekawym natomiast jest fakt, skąd biorą się skojarzenia i jakie są mechanizmy ich powstawania. Co sprawia, że w konkretnej sytuacji przychodzi na myśl takie a nie inne skojarzenie? Kontekstem dla procesów myślenia, może być sytuacja, która jest podobna do innych zdarzeń z przeszłości. Umysł uzupełniać ją może o okoliczności towarzyszące, takie jak nasze potrzeby czy postawiony cel. Ciągle otacza nas mnóstwo bodźców, więc do analizy dostarczane jest wiele danych, które mogą aktywować skojarzenia. Z punktu widzenia komputerów, możliwe byłoby przeszukiwanie pamięci i doszukiwanie się podobnego miejsca, które odpowiadałoby sekwencji skojarzeń. Jest to jednak wykluczone, ponieważ umysł porusza się w nieskończonej przestrzeni i na bieżąco tworzy nowe skojarzenia, które uruchamiają poprzednie myśli lub kontekst otoczenia. Dzieje się to bardzo szybko, dzięki temu możemy stwierdzić, że niezaangażowane są w ten proces żadne algorytmy czy programy. Połączone w mózgu neurony pobudzane są bezpośrednio treścią (danymi i ich kombinacjami). Jak wiadomo mózg jest zbudowany z sieci połączonych ze sobą neuronów, których stan aktywności

jest zależny od stopnia ich pobudzenia. Wszystkie neurony działają równolegle w dużym stopniu niezależnie od siebie. Zależne są od swojego wcześniejszego stanu oraz aktywności innych neuronów z nimi połączonych za pośrednictwem połączeń synaptycznych. Co więcej w procesach myślowych, kontekst zmienia się bardzo dynamicznie i na skojarzenie wpływ ma tylko ten najbardziej aktualny. Zostało udowodnione, że podstawą ludzkich skojarzeń jest podobieństwo oraz kolejność. Skojarzenia nie działają wspak, większość neuronów nie jest wstanie przekazywać sygnału dwukierunkowo, czy bez osiągnięcia stanu aktywacji. W procesie przetwarzania danych istotne znaczenie ma kolejność i czas pobudzenia, tłumienia i aktywowania powiązanych neuronów. Wynika z tego, że pozornie ten sam kontekst zewnętrzny, może spowodować inną reakcję ze względu na inny kontekst wewnętrzny. Napływ takich samych danych, ale w innej kolejności czy w różnych odstępach czasu ma fundamentalne znaczenie dla procesów skojarzeniowych. Bardzo dobrze prezentuje to przykład: wypowiedzenie słów „panna młoda" i „młoda panna" prowadzi zwykle do zupełnie innych skojarzeń [7].

3.3.2. Definicja asocjacji

Asocjacja według słownika języka polskiego jest to „*ugrupowanie, zbiorowisko, zespół;* w psychologii*: łączenie ze sobą wrażeń i wyobrażeń"* [9]. Definicja sama w sobie jest dość ogólna i pozwala na szeroką interpretację. Natomiast czynność kojarzenia została już uściślona i ogranicza się do stwierdzenia, iż kojarzyć to znaczy „*powodować połączenie, związek czegoś lub kogoś; łączyć wrażenia, wyobrażenia tak, że pojawienie się jednych powoduje uświadomienie sobie innych"* [9]. Kojarzenie jest aktywnym i dynamicznym procesem zależnym od uformowanej wcześniej wiedzy, czasu, podobieństwa, kolejności, reaktywności elementów systemu czy kontekstu. Proces ten polega na kontekstowym wiązaniu ze sobą danych, ich kombinacji i układów poprzez tworzenie nowych lub wzmacnianie istniejących połączeń synaptycznych w grafie aktywnych neuronów. Kontekst powiązania zależy od uprzednio uformowanej wiedzy. W danym systemie skojarzeniowym, napływające dane aktywują istniejące w nim neuronowe reprezentacje. Samo wywoływanie skojarzeń (czy też przypominanie sobie czegoś) zachodzi na skutek stopniowej aktywacji neuronów, reprezentujących pewne kombinacje i układy danych. Neurony mogą aktywować się w różnej kolejności i w różnym stopniu, w zależności od oddziałujących na nie bodźców oraz kontekstu wcześniejszych pobudzeń.

Za definicję skojarzeniowości systemu możemy uznać zdolność do dynamicznego, plastycznego, aktywnego i grafowego wiązania ze sobą danych i ich kombinacji w taki sposób, by odwzorowane było ich podobieństwo, kolejność, zależność, względna równoczesność i kontekst ich występowania. Skojarzeniowość jest kluczową cechą inteligentnych biologicznych systemów przetwarzających dane, która pozwala na formowanie aktywnej i dynamicznej wiedzy [7].

3.3.3. Relacje asocjacyjne

Pomiędzy danymi i ich układami mogą zachodzić różne relacje, które mają zwykle pośredni lub bezpośredni wpływ na nasze rozumowanie. Jeśli dwa obiekty są do siebie podobne, zazwyczaj nam się ze sobą kojarzą, mimo że nigdy nie uczyliśmy się o rozpoznawaniu takich podobieństw. Nasz umysł automatycznie je rozpoznaje, ponieważ wspólne cechy podobnych obiektów, aktywują te same grupy neuronów. Wystarczy później, że te obiekty występują w pewnym sąsiedztwie, lub pewniej kolejności a system skojarzeniowy je ze sobą wiąże. Zauważono, że fizyczne połączenia między neuronami są do siebie podobne, natomiast różnią się one od strony funkcjonalnej. W każdym grafie asocjacyjnym możemy wyróżnić kilka rodzajów relacji [7]:

- asocjacyjnego podobieństwa (ASIM),
- asocjacyjnego następstwa (ASEQ),
- asocjacyjnego kontekstu (ACON),
- asocjacyjnego definiowania (ADEF),
- asocjacyjnego tłumienia (ASUP).

Powiązania asocjacyjnego podobieństwa (ASIM – *ang. associative similarity*) łączą podobne, bliskie dane i ich kombinacje, w sensie wybranej metryki. Im bliższe lub bardziej podobne wartości bodźców reprezentują, tym bardziej wzmacniają połączenie między sobą (poprzez równoczesne pobudzanie i aktywowanie zwiększa się waga połączenia). W naturalnych sieciach neuronowych połączenie dwóch neuronów jest jednokierunkowe, a w sieciach sztucznych może być jednostronne lub wielostronne. Połączenia ASIM mają zawsze charakter pobudzający.

Powiązania asocjacyjnego następstwa (ASEQ *ang. associative sequence*) łączą dane, które następują po sobie chronologicznie. Powiązania te powstają na skutek wzmocnienia połączeń między neuronami, które wielokrotnie ulegają kolejnej aktywacji. Jeśli pewne bodźce występują po sobie lub w niewielkim odstępie czasu i aktywują one neurony ze sobą połączone, wtedy dochodzi do wzmocnienia połączenia między takimi neuronami, i utrwalone zostają powstałe chronologiczne zdarzenia. Ten typ powiązań jest asocjacją jednokierunkową i pozwala nam na odtwarzanie melodii, czy ciągu usłyszanych słów. Powiązania typu ASEQ mają zawsze charakter pobudzający i odgrywają ważną rolę w procesie uczenia się i formowania wiedzy.

Powiązania asocjacyjnego kontekstu (ACON - *ang. associative context*) łączą dane, ale zwykle nie prowadzą samodzielnie do aktywacji neuronu, lecz umożliwiają jego aktywację w niedalekiej przyszłości. Umożliwiają one uwzględnienie kontekstu wcześniej aktywowanych neuronów przez te aktualnie pobudzane. Powiązania typu ACON mogą mieć charakter hamujący lub pobudzający.

Powiązania asocjacyjnego definiowania (ADEF - *ang. associative defining*) pojawiają się, gdy przekazywane są dane z receptorów lub neuronów przekazujących dane wejściowe, reprezentują niektóre kombinacje lub układy. Powiązania te powstają w wyniku

równoczesnego oddziaływanie wielu receptorów. Mogą łączyć wiele danych receptorycznych, ich zakresów i kombinacji, definiując nowe kombinacje, które mogą reprezentować pewne obiekty lub zdarzenia.

Powiązania asocjacyjnego tłumienia (ASUP - *ang. associative suppresion*) łączą dane w celu ich wyostrzenia, dyskryminacji, zwiększenia kontrastu lub wzajemnego tłumienia. Połączenia te uniemożliwiają dublowanie reprezentacji podobnych i najczęstszych kombinacji. Asocjacyjne tłumienie ma za zadanie skontrastować i wyostrzyć bodźce wejściowe oraz uniemożliwić automatyczne reprezentowanie danych tego samego rodzaju, przez te same układy skojarzeniowe [7].

3.4. Struktury danych AGDS i AANG

Sztuczne systemy skojarzeniowe (AAS – *ang. Artificial Associative Systems*) odwzorowują i modelują biologiczne systemy skojarzeniowe, czyli ludzki mózg i zachodzące w nim procesy. Są udoskonalone o dodatkowe mechanizmy i elementy, które zwiększają efektywność rozwiązania poszczególnych zadań.

Asocjacyjna grafowa struktura danych (AGDS – *ang. Associative Graph Data Structure*) to graf, który przechowuje dane, ich kombinacje oraz relacje między nimi zachodzące. Struktura grafu nie odwzorowuje czasu, jest ona statyczna, podobnie jak bazy danych. Podstawowe połączenia asocjacyjne występujące w grafie to ASIM, ASEQ i ADEF. Ze względu na brak wspomnianego pojęcia czasu, grafy te nie używają bardziej zaawansowanych mechanizmów asocjacyjnych jak połączenia ASUP i ACON. Dane przechowywane są jako węzły, a zależności pomiędzy nimi reprezentowane są przez krawędzie. Wartość połączenia dwóch węzłów, wyrażana jest za pomocą wagi nadawanej poszczególnej krawędzi. Jedną z większych zalet struktury jest praktyczny brak redundancji, wszystkie dane są unikalne i uporządkowane. Dzięki skorelowaniu poszczególnych obiektów, operacje wyszukiwania obiektów podobnych czy różnych są szybkie, a same struktury możemy nazwać oszczędnymi. Grafy te idealnie rozwiązują problemy dotyczące redundancji czy wydajności, nie należy ich natomiast stosować do przechowywania zupełnie niepowiązanych ze sobą danych [7].

Formalnie graf AGDS można przedstawić, jako uporządkowaną siódemkę [7]:

$$AGDS = (VV, VR, VS, VC, ESIM, ESEQ, EDEF)$$

gdzie *VV, VR, VS, VC* to zbiory wierzchołków, a *ESIM, ESEQ, EDEF* to zbiory krawędzi, które można zdefiniować następująco:

VV – zbiór wierzchołków reprezentujących pojedynczą wartość (*ang. value vertex*);
VR – zbiór wierzchołków reprezentujących przedział wartości (*ang. range vertex*);
VS – zbiór wierzchołków reprezentujących podzbiór wartości (*ang. subset vertex*);

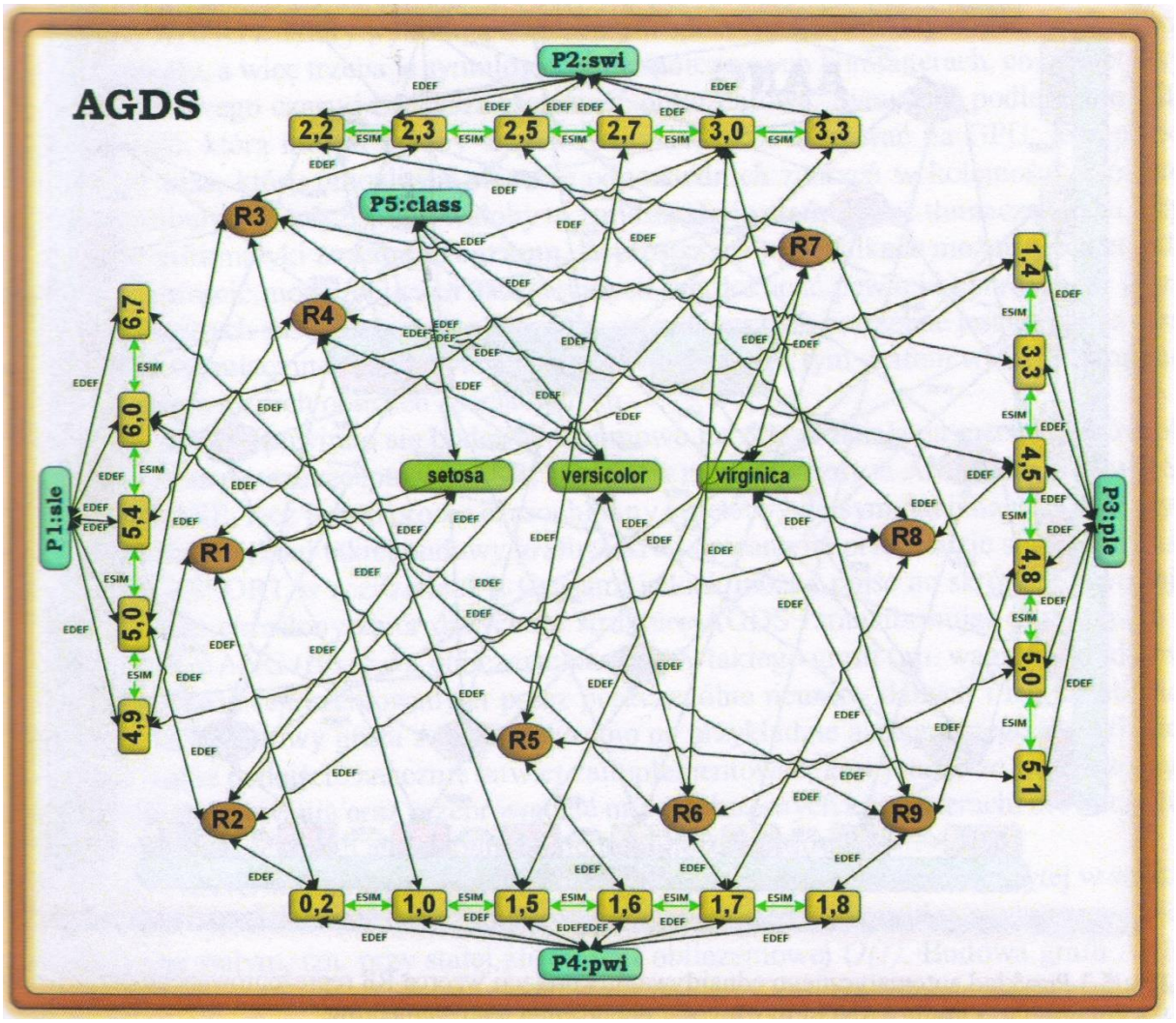
VC – zbiór wierzchołków reprezentujących kombinację wartości (*ang. combination vertex*);
ESIM – zbiór krawędzi nieskierowanych (*ang. undirected edge*) łączących asocjacyjnie podobne wierzchołki (*ang. similarity edge*);
ESEQ – zbiór krawędzi skierowanych (*ang. directed edges*) łączących asocjacyjnie następne wierzchołki (*ang. sequence edge*);
EDEF – zbiór krawędzi dwustronnie skierowanych (*ang. bidirected edges*) łączących wierzchołek definiujący z wierzchołkiem definiowanym (*ang. defining edge*) w taki sposób, że inna etykieta (waga) określa przejście z od jednego do drugiego wierzchołka.

Poniżej zaprezentowany został przykład, w którym prosty zbiór danych został przekonwertowany do struktury AGDS. Mimo że dla tak małego zbioru tabelarycznego, struktura AGDS wydaje się bardziej skomplikowana, zauważymy od razu, jakie wartości charakteryzują dany zbiór. Co więcej, patrząc na strukturę AGDS przez pryzmat optymalizacji wyszukiwania, łatwo dostrzeżemy potencjalnie mniejszą złożoność obliczeniową.

WZORCE IRIS

	sle	swi	ple	pwi	class
R1	4,9	3,0	1,4	0,2	setosa
R2	5,0	3,3	1,4	0,2	setosa
R3	5,0	2,3	3,3	1,0	versicolor
R4	5,4	3,0	4,5	1,5	versicolor
R5	6,0	2,7	5,1	1,6	versicolor
R6	6,7	3,0	5,0	1,7	versicolor
R7	6,0	2,2	5,0	1,5	virginica
R8	4,9	2,5	4,5	1,7	virginica
R9	6,0	3,0	4,8	1,8	virginica

Rys.8. Tabelaryczna struktura danych zbioru „Iris”[7]



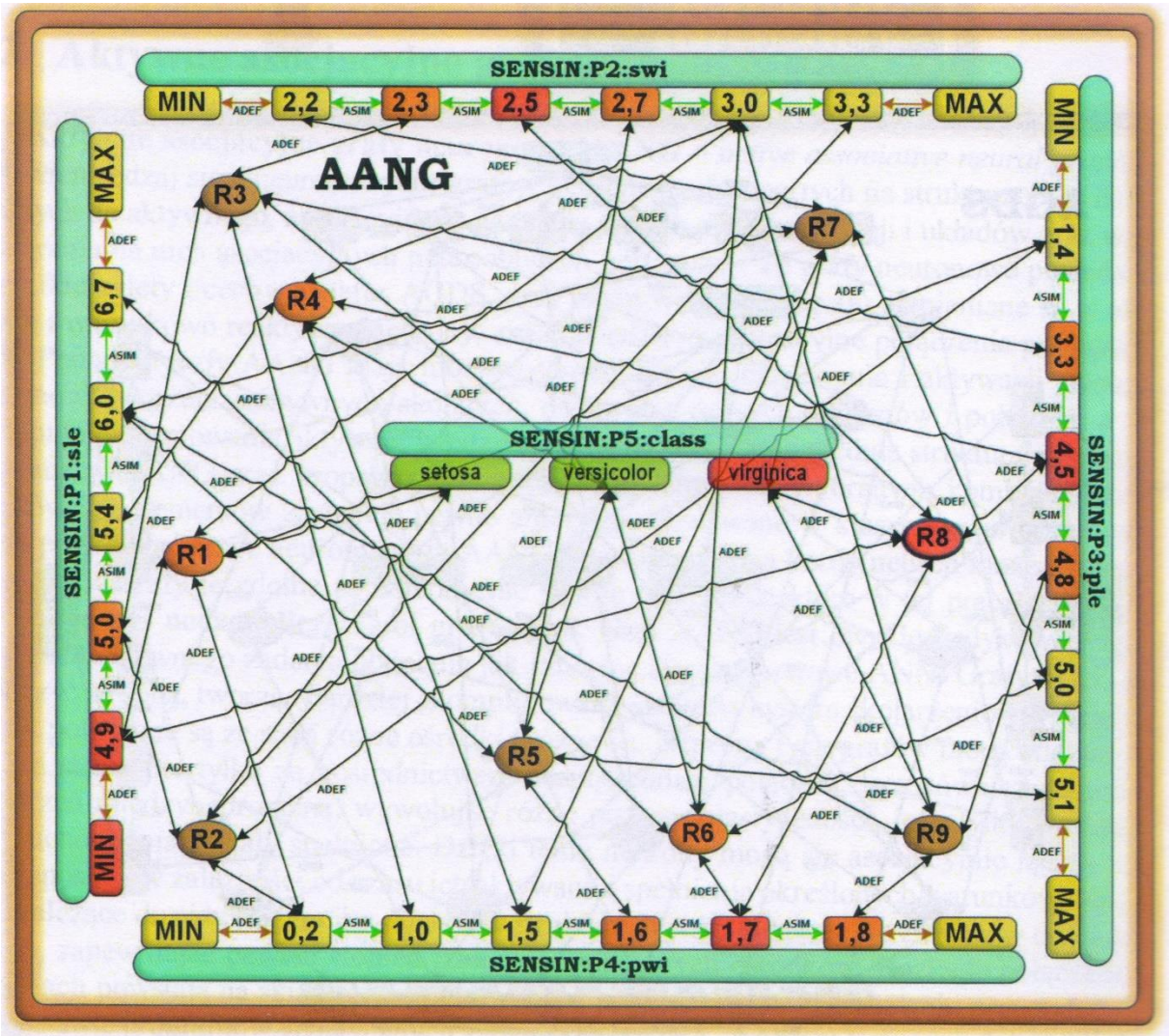
Rys.9. Struktura danych zbioru „Iris" przedstawiona za pomocą grafu AGDS [7]

Drugą ciekawą strukturą skojarzeniową jest AANG (*ang. Active Associative Neural Graphs*). Grafy te oparte są bezpośrednio na strukturach AGDS, są też pewnym rodzajem sieci neuronowych. Węzły grafu zamienione są w neurony, a krawędzie w asocjacyjne połączenia między nimi. Zawierają one wszystkie cechy i zalety struktur AGDS, przy czym uwzględniają pojęcie czasu. Dzięki temu aktywne asocjacyjne grafy neuronowe mają możliwość pobudzania, hamowania i aktywacji swoich neuronów. Mogą tworzyć nowe neurony i skojarzenia między nimi, a co za tym idzie powiększać wiedzę skojarzeniową. Poprzez zastosowanie odpowiedniej struktury i parametrów, możliwe jest zastąpienie pewnych algorytmów stosowanych w klasycznej informatyce.
Aby formalnie opisać grafy AANG należy jeszcze wyszczególnić:

SENSIN – wejście sensoryczne (*ang. sensory input*);
VN – neuron wartości (*ang. value neuron*) - reprezentuje wartości powiązań z danym SENSIN -em;

SN – asocjacyjny neuron wzorca (*ang. sample neuron*) - reprezentuje dane definiowane przez VNy i inne SNy.

Rysunek 10 prezentuje strukturę AANG utworzoną z prostej struktury tabelarycznej z zewnętrznie aktywowanym wzorcem R8 i jego korelacjami.



Rys.10. Struktura danych zbioru „Iris" przedstawiona za pomocą grafu AANG [7]

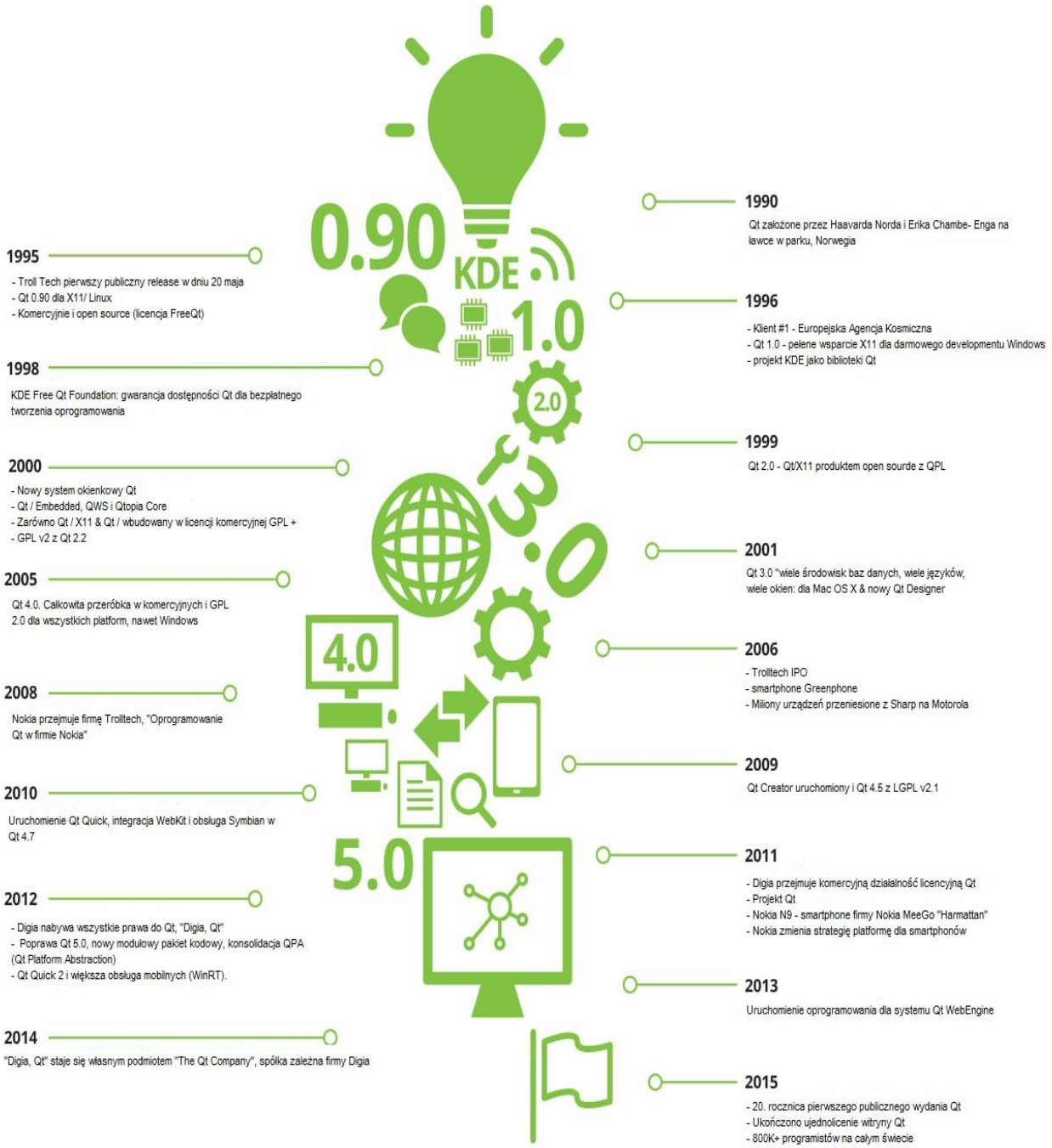
4. Narzędzia programistyczne Qt

Qt jest to zestaw przenośnych bibliotek i narzędzi programistycznych przeznaczony dla języków C++, QML i Java. Jego podstawowym elementem są klasy umożliwiające tworzenie interfejsów graficznych. Od wersji 4.0 oprogramowanie zawiera również narzędzia do tworzenia programów konsolowych i serwerów. Środowisko Qt pozwala na development aplikacji desktopowych, mobilnych oraz dla systemów wbudowanych. Wspierane jest przez platformy takie jak: Windows, Linux, OS X, VxWorks, QNX, Android, iOS oraz BlackBerry.

Biblioteki Qt charakteryzuje pełna obiektowość. Zawierają one wiele zaawansowanych technologii programowania graficznego interfejsu użytkownika, takich jak: mechanizm sygnałów i slotów, automatycznego rozmieszczania widżetów, zhierarchizowany system obsługi zdarzeń. Dodatkowo, w ich składzie znajdują się moduły obsługi procesów, sieci, grafiki komputerowej (OpenGL), baz danych (SQL), lokalizacji, języka XML czy wielowątkowości[10]. W skład Qt wchodzą takie specjalistyczne narzędzia programistyczne jak:

- MOC *(Meta Object Compiler)* – preprocesor, który na podstawie plików nagłówkowych generuje pliki źródłowe,
- qmake – program do zarządzania procesem kompilacji,
- UIC (*User Interface Compiler)* – kompilator plików *.ui.

Dodatkowo Qt wyposażony jest w Qt Designer, czyli aplikację graficzną do definiowania graficznego interfejsu użytkownika oraz Qt Creator – zintegrowane środowisko programistyczne.



Rys.11. Historia Qt [11]

4.1. Qt Creator i Qt Designer

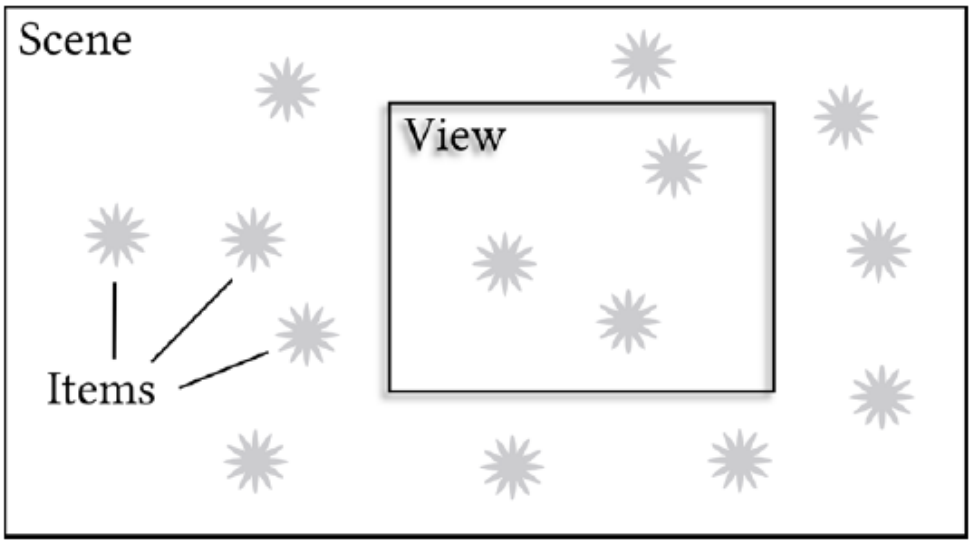
Rys.12. Ikona programu Qt

Qt Creator jest zintegrowanym środowiskiem programistycznym (IDE, ang. *Integrated Development Environment*) wyposażonym w debugger, profilowanie, funkcję podświetlania składni oraz narzędzie do automatycznego uzupełniania m.in. nazw funkcji i zmiennych (IntelliSense). Służy on do tworzenia, modyfikacji, testowania i konserwacji oprogramowania. Qt Creator używa kompilatora MinGW na Windowsie oraz kompilatora G++ na Linuksie i OS X. Obsługuje systemy kontroli wersji: Bazaar, ClearCase, CVS czy git.

Qt Desinger to aplikacja graficzna, która służy do projektowania i budowania interfejsów graficznych użytkownika *(GUI).* Przy użyciu QtWidgets, czyli modułu z elementami graficznymi możliwe jest szybkie i łatwe komponowanie i dostosowywanie tworzonych okien. Elementy graficzne powiązane są z dynamicznym zachowaniem z wykorzystaniem mechanizmu sygnałów i slotów. Wszystkie właściwości elementów graficznych mogą być edytowane w designerze jak i bezpośrednio w kodzie.

4.2. Graphics View

Graphics View jest to framework wykorzystywany przy tworzeniu skomplikowanych aplikacji wizualnych. Ułatwia zarządzanie dużą ilością własnych elementów grafiki 2D, służy do ich wizualizacji, umożliwia ich obrót czy powiększanie. Złożony jest z trzech komponentów: *QGraphicsView*, który odnosi się do widoku (ang. *View*), *QGraphicsScene* (ang. *Scene*), które referuje do sceny oraz *QGraphicsItem* stosuje się do reprezentacji poszczególnych obiektów.



Rys.13. Komponenty QGraphics[17]

Scena jest obszarem, w którym przechowywane są wszystkie obiekty. Jest ona odpowiedzialna za zapewnienie szybkiego interfejsu, zarządzanie stanem obiektu (np. obiekt wybrany, obiekt niewidoczny, obiekt z fokusem) czy doprowadzanie eventów do obiektów. *QGraphicsView* dziedziczy po *QWidget*, co za tym idzie za jego pomocą tworzony jest interfejs graficzny. Ma on możliwość podglądu całej sceny lub jej fragmentów. Architektura Graphics View umożliwia również manipulację współrzędnymi sceny. Ostatnim, najmniejszym i najliczniejszym komponentem Graphics View jest *QGraphicsItem.* Klasa ta reprezentuje pojedyncze obiekty, które mogą być dziedziczone po prostokącie (*QGraphicsRectItem*) lub po elipsie (*QGraphicsEllipseItem*). Każdy obiekt wyposażony jest w reakcję na ruchy myszką, polecenia z klawiatury czy detekcję kolizji[12].

4.3. Język programowania C++

C++ jest językiem programowania ogólnego przeznaczenia. Został zaprojektowany przez Bjarne Stroustrupa jako rozszerzenie języka C o silną statyczną kontrolę typów i obiektowe mechanizmy abstrakcji danych. Swoją popularność zyskał w latach 90 XX wieku. C++ charakteryzuje się dużą wydajnością kodu wynikowego, bezpośrednim dostępem do funkcji systemowych i zasobów sprzętowych, łatwością tworzenia i korzystania z bibliotek. Jest językiem wieloparadygmatowym, co oznacza, że jednocześnie można w nim stosować różne style programowania (proceduralne, obiektowe, generyczne). Umożliwia abstrakcję danych i bezpośrednie zarządzanie wolną pamięcią.

Język C++ jest uznawany za jeden z najlepszych języków programowania przy profesjonalnym tworzeniu gier. Gry komputerowe pełne są obiektów, np. statków kosmicznych

czy postaci, które w naturalny sposób reprezentowane są jako dane i metody. W grach komputerowych duże znaczenie ma wydajność. C++ kompiluje się bezpośrednio na kod assembly, stąd instrukcje dla CPU są bardziej bezpośrednie, niż przy innych językach obiektowych jak: Python, czy Java. Dodatkowo C++ jest językiem dojrzałym, z dużą ilością zaawansowanych bibliotek [13].

5. Projekt gry komputerowej

5.1. Założenia ogólne

Podstawowym założeniem realizowanego projektu było stworzenie prostej i jednocześnie atrakcyjnej gry komputerowej. Prostota ta miała polegać na odpowiednim uporządkowaniu i zapewnieniu przejrzystości wykorzystywanego kodu, jak też minimalizacji zawiłych relacji między elementami programowania obiektowego. Natomiast głównym elementem stanowiącym o atrakcyjności, mieli być przeciwnicy z zaimplementowanymi systemami skojarzeniowymi, umożliwiającymi osiągnięcie inteligentnych reakcji w różnych kontekstach w trakcie gry. Grafowy model asocjacyjny odpowiedzialny był za optymalność działania oraz szybkość podejmowanych decyzji podczas działania programu.

Chcąc by powyższe założenia zostały spełnione, jeszcze przed rozpoczęciem pisania kodu, należało określić listę kluczowych wymagań dla aplikacji. Stanowią one, nie tylko wyznacznik kierunku przebiegu dalszych prac, ale opisują również zasady dobrego stylu programowania, które z dużym naciskiem zostały wprowadzone podczas implementacji gry „CleverWorm". Całość tych wymagań można zestawić następująco:

- Wszystkie stałe wielokrotnie stosowane w programie, powinny być dostępne, bez konieczności formowania sztucznych hierarchii klas. Ich nazewnictwo powinno być z góry ściśle określone i bezwzględnie stosowane.
- Aplikacja nie może być uzależniona od plików zewnętrznych. Program powinien być odseparowany od danych wejściowych.
- Każdy moduł powinien spełniać ściśle określony zestaw zadań, do których został powołany. Nie jest możliwa sytuacja, w której, klasa znajduje się w stanie, w którym odpowiedzialność za niewłaściwe funkcjonowanie leży po stronie użytkownika lub innej klasy.
- Obowiązuje zasada pojedynczej odpowiedzialności: „*Żadna klasa nie może być modyfikowana z więcej niż jednego powodu*" [14].
- Obowiązuje zasada otwarte - zamknięte: „*Składniki oprogramowania (klasy, moduły, funkcje itp.) powinny być otwarte na rozbudowę, ale zamknięte dla modyfikacji*" [14].
- Obowiązuje zasada odwracania zależności, która mówi, że moduły wysokopoziomowe nie powinny być zależne od modułów niskopoziomowych, tylko od abstrakcji [14].
- Kod aplikacji powinien zawierać możliwie mało zależności pomiędzy swoimi składnikami (klasami), pozostając przy tym elastycznym.
- Każda funkcja spełnia jedno, z góry określone zadanie, na które wskazuje jej nazwa.

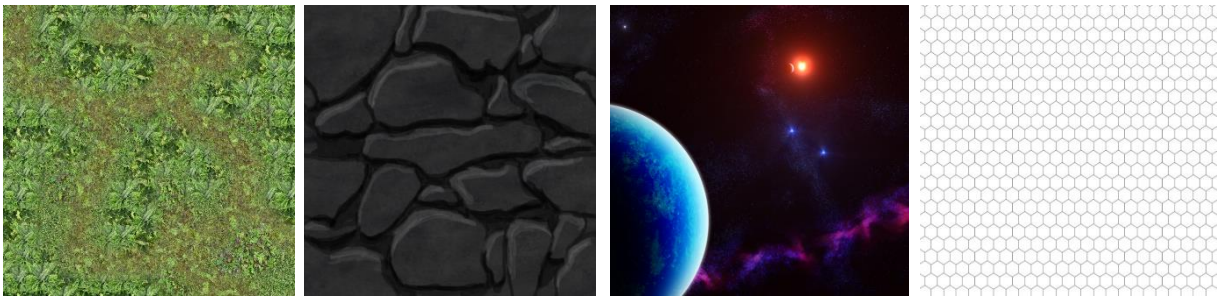
- Nazwa klasy, funkcji czy przestrzeni nazw musi zostać umieszczona w odpowiednim kontekście, tak by niosła ze sobą wystarczającą treść.
- Kod musi zostać dobrze napisany jak i utrzymywany. Każda wprowadzana zmiana musi być spójna. Stosujemy się do zasady „*Pozostaw obóz czyściejszym niż go zastałeś*” [15].
- Kod powinien zostać sformatowany, aby zapewnić dobrą komunikację.
- Zajętość pamięci operacyjnej jest minimalna, program nie powinien wczytywać nadmiarowych plików graficznych bez wyraźnej potrzeby. Pamięć obiektów tworzonych dynamicznie jest każdorazowo zwalniana.

Każda z wymienionych zasad ma na celu polepszenie jakości kodu. Minimalizowany jest jego poziom skomplikowania a maksymalizowana jest podatność na zmiany funkcjonalności. Spełnienie powyższych wymagań powinno zaowocować w zaimplementowaniu obiektowej, niezależnej i wydajnej gry typu „Snake”. Gra składać się będzie z kilku złożonych asocjacyjnych struktur grafowych, które będą stanowić jej silnik oraz dodatkowych elementów dbających o efekty funkcjonalne i graficzne.

5.2. Elementy gry

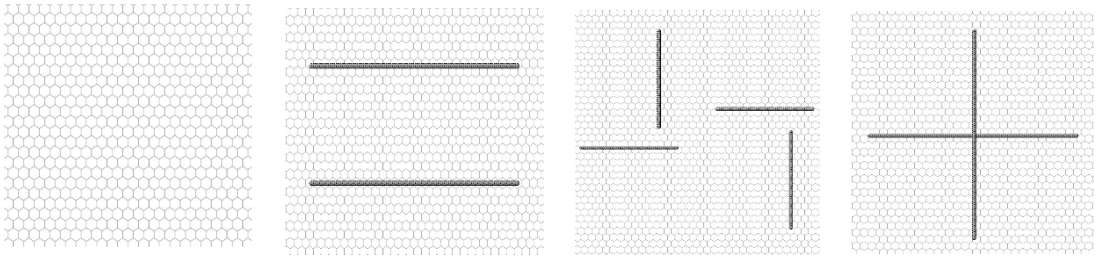
Przed przedstawieniem dalszej części projektu gry oraz jej implementacji istotne jest przybliżenie niektórych pojęć, które w dalszej części pracy pojawiać się będą w różnym kontekście. Poniżej zostaną pokrótce scharakteryzowane główne elementy gry „*Clever Worm*”.

Plansza – to miejsce rozgrywki, posiadające z góry określone wymiary, których przekroczenie skutkuje utratą życia. W zależności od wgranego trybu symbolizuje krainę, w której odbywa się rozgrywka.



Rys.14. Plansze dostępne w grze „CleverWorm”

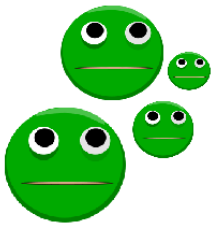
Mur – to statyczna przeszkoda o określonym kształcie zależnym od wybranego trybu. Kolizja z nią skutkuje utratą życia



Rys.15. Układy muru dostępne w grze „CleverWorm”

Rys.16. Grafika reprezentująca pożywienie

Pożywienie – to logiczna struktura danych, którą możemy utożsamić z posiłkiem. Pozyskanie go zwiększa ilość punktów przypisanych do drużyny jak i rozmiar zawodnika. Posiada takie atrybuty jak miejsce występowania i wielkość. Jest zawsze koloru żółtego o okrągłym kształcie. Wielkość kuli pokarmowej jest liniowa w stosunku do możliwych do pozyskania punktów. Ilość pożywienia jest określona i stała dla konkretnych poziomów.

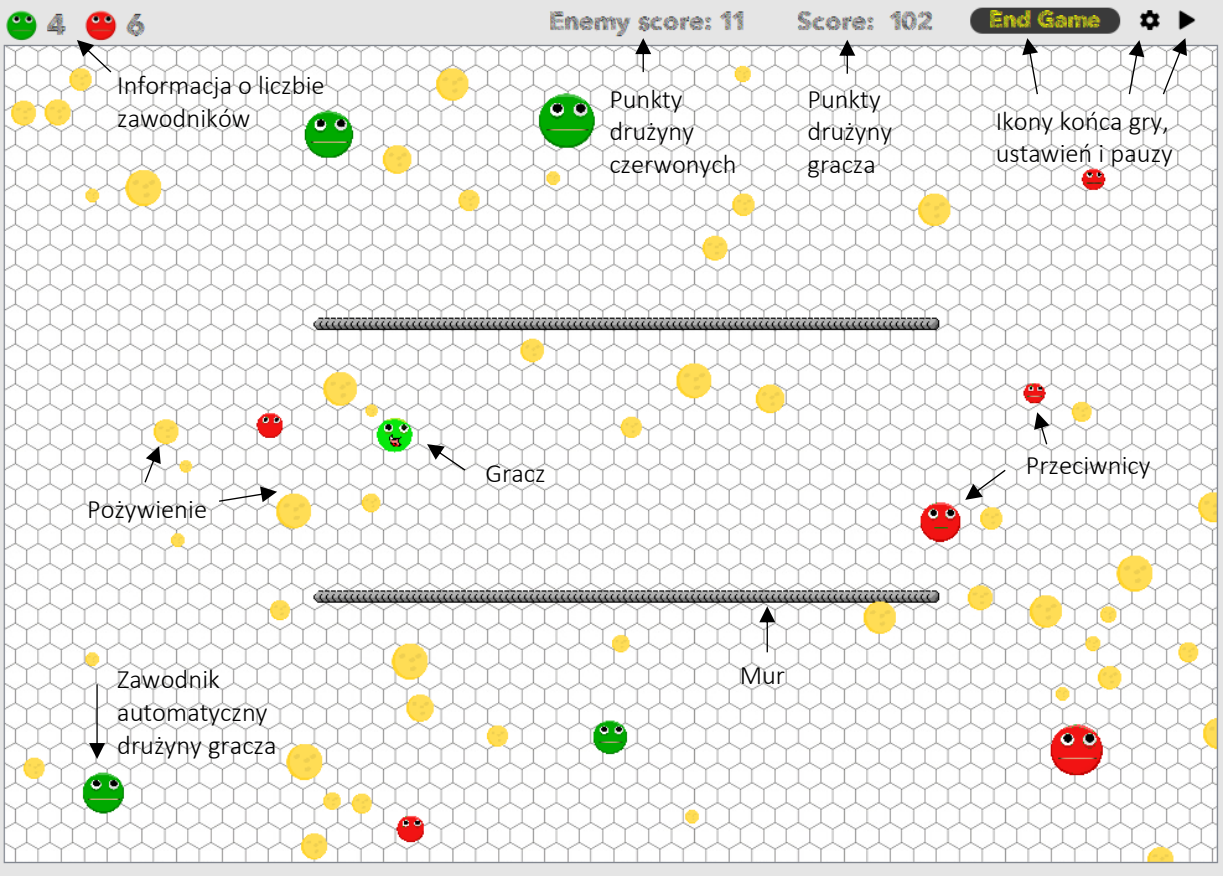


Rys.17. Grafika reprezentująca automatycznych zawodników

Worm – to zawodnik sterowany za pomocą mechanizmów skojarzeniowych. Należy do jednej z dwóch drużyn: zielonej lub czerwonej. Jego celem jest zdobywanie pożywienia oraz unikanie zagrożeń. Porusza się liniowo odgórnie określoną prędkością. Charakteryzują go takie atrybuty jak kolor, wielkość oraz położenie. Liczba zawodników definiowana jest dla konkretnego poziomu

Rys.18. Grafika Reprezentująca gracza

Player – to zawodnik sterowany przez gracza komputerowego. Zawsze członek drużyny zielonych. Kierowany jest za pomocą myszki. Wraz z pozyskiwanym pokarmem jego rozmiar się zwiększa.



Rys.19. Gra „Clever Worm" i jej elementy

5.3. Zasady gry

Celem gry jest zdobywanie pożywienia i eliminacja przeciwników. Każdorazowy kontakt z ikoną jedzenia, skutkuje jego spożyciem i jednocześnie wzrostem masy. W grze rywalizują ze sobą dwie drużyny, zielonych i czerwonych, gracz komputerowy zawsze należy do drużyny zielonych. Gdy dwóch zawodników z różnych drużyn się ze sobą spotka, o wygranej w pojedynku decyduje wielkość – zwycięzca musi być o 30% większy od rywala. Każdorazowe zjedzenie rywala, skutkuje otrzymaniem punktów w wysokości trzykrotności wielkości masy przeciwnika. Jeśli zawodnik zostanie zjedzony przez gracza komputerowego, otrzymuje on również trzykrotność punktów. Gdy pojedynek ma miejsce wśród jednej drużyny, za zjedzenie zawodnika z własnego teamu otrzymuje się punkty karne (minusowe) w tej samej wysokości. Jeśli śmierć zawodnika nastąpi z innego powodu np. przekroczenia planszy, lub zderzenia z murem, również naliczane są punkty minusowe. Punkty liczone są sumarycznie dla drużyny (automatycznej – czerwonej, automatycznej z 1 graczem – zielonej) a nie osobno dla zawodnika. Ilość zawodników w drużynie jest stała i określona dla poziomu, zawsze znana graczowi. Pożywienie generowane jest automatycznie, jego ilość jest również stała dla poziomu. Po zjedzeniu obiektu tworzony jest inny, o losowej wielkości w losowym miejscu na planszy. W grze dostępnych jest 5 poziomów trudności, które odblokowywane są po uzyskaniu odpowiedniej liczby punktów przez drużynę gracza (zieloną). Poziomy te charakteryzują się

różną licznością drużyny rywali, własnej drużyny oraz licznością pożywienia. Gra „Clever Worm" charakteryzuje się 4 planszami tematycznymi oraz 4 różnymi formami muru ograniczającego.

Funkcjonalności dostępne dla gracza komputerowego:

- wczytanie nowej gry;
- zastopowanie rozgrywki w dowolnym momencie i powrót, po wciśnięciu przycisku;
- rozpoczęcie nowej gry, po przegranej;
- sterowanie własnym zawodnikiem za pomocą myszki komputerowej;
- uruchomienie funkcji „speed" po znalezieniu odpowiednio dużego pożywienia;
- wybór planszy tematycznej;
- wybór formacji muru.

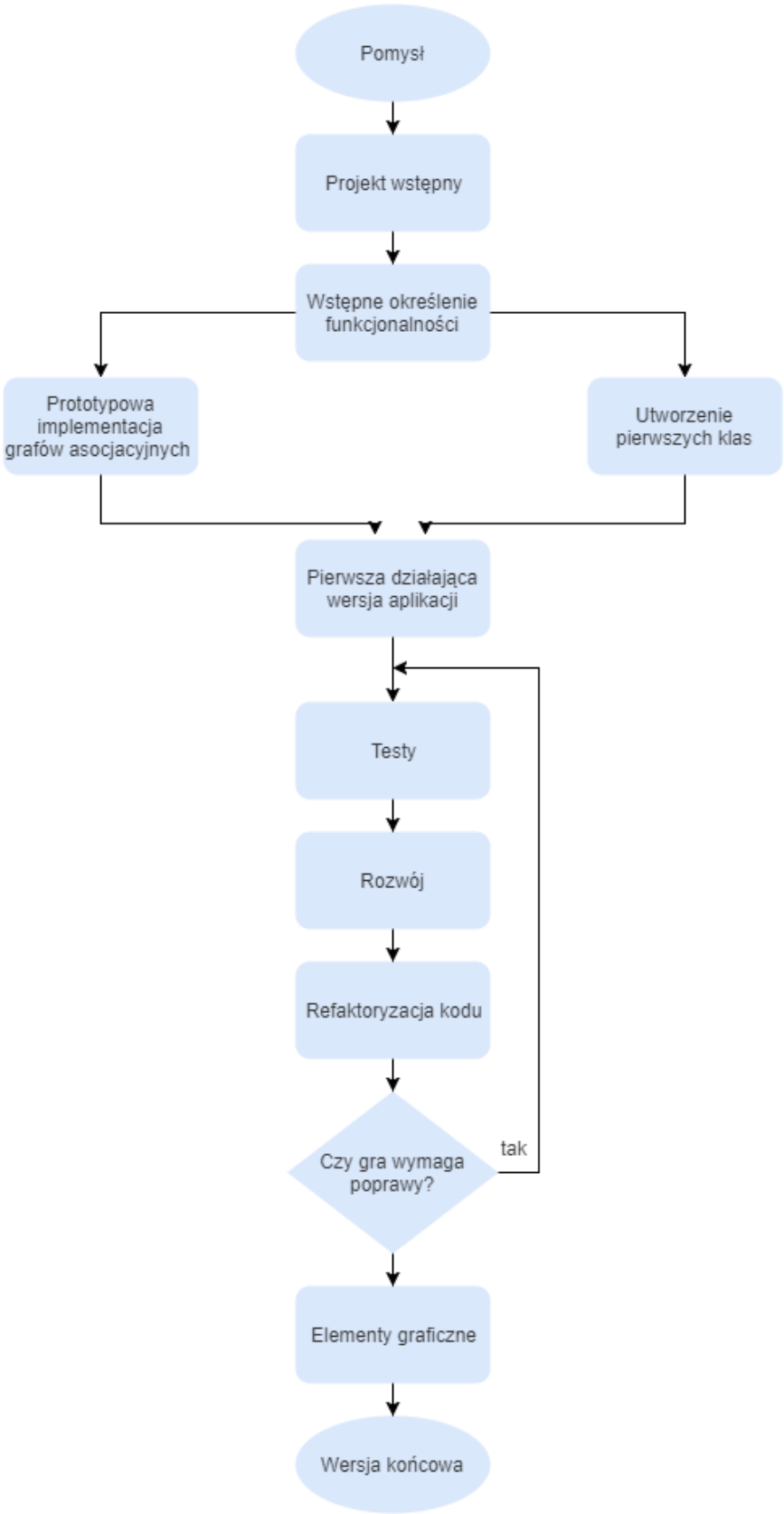
6. Architektura systemu

6.1. Cykl projektowy

Inżynieria oprogramowania najogólniej może być zrozumiana, jako sztuka pisania programów. Działania w jej obrębie rozpoczynają się od analizy wymagań. Następnie poprzez projektowanie i implementację tworzymy produkt. Później, aż do wdrożenia i utrzymywania aplikacji, jesteśmy za niego odpowiedzialni. Bazą dla dobrego programu komputerowego jest spełnienie wymagań klienta, które niezwykle często są niejasne, rozmyte i zmienne w czasie. Powoduje to, że architektura systemu musi być elastyczna, a prace projektowe logicznie uporządkowane. Metodyka strukturalna zapewnia, że każde wymaganie klienta pokryte jest kodem i strukturą zapewniającą realizację tego wymagania.

Typowy cykl projektowy zakłada takie fazy jak: strategia, analiza, projektowanie, implementacja wdrożenie oraz utrzymanie. Klasyczną jego realizacją jest podejście „wodospadowe", w którym elementy wykonywane są kolejno po sobie. Podejście to pozwala na wygodną ocenę czasu oraz kosztów projektu, jednak w dużej mierze uzależnia efekt końcowy od początkowego etapu analizy i projektowania. Innym obecnie bardzo popularnym podejściem określającym budowę architektury jest metodyka zwinna (ang. *agile software development*). Opiera się ona na przyrostowym modelu budowania aplikacji. Zakłada częste i stosunkowo wczesne wydawanie nowych prototypów programu. Na początku określany jest ogólny zbiór wymagań i co pewien ustalony czas (ang. *sprint*) wybrana część z nich zostaje implementowana. Metodyka ta charakteryzuje się dużą zmiennością, pozwala na bieżąco korygować i kontrolować rozwój aplikacji, bez konieczności gruntownej przebudowy całego projektu. Do jej minusów należą duże ilości wprowadzanych zmian i poprawek do wcześniejszych uzgodnień.

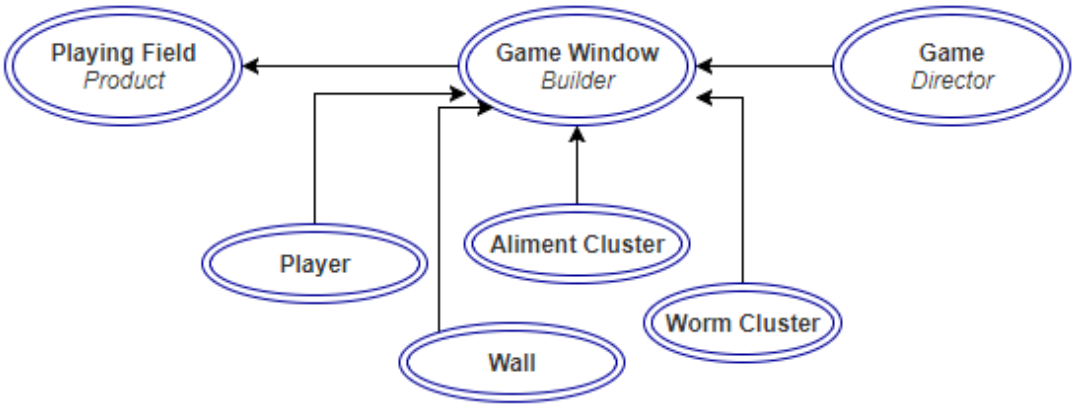
Ze względu na niedużą wielkość projektu oraz niekonkretne początkowe wymagania, projekt gry komputerowej zdecydowano się realizować za pomocą metodyki zwinnej. Ostateczna koncepcja gry „*Clever Worm*" wykrystalizowała się po kilku miesiącach pracy opisanych przez powyżej przedstawione sprinty. Od samych początków największy nacisk kładziony był na implementację grafowych mechanizmów skojarzeniowych sterujących automatycznymi zawodnikami. Następnie wprowadzono postać gracza zdalnego oraz funkcje umożliwiające analizę kontekstową i wnioskowanie. Jednym z ostatnich działań było uatrakcyjnienie gry o dołożenie drużyn z zawodnikami grającymi przeciwko sobie i walczącymi o wspólne pożywienie. Finalnym elementem było dopracowanie elementów graficznych. Opisane powyżej funkcjonalności zostały opisane na poniższym diagramie (rysunek 20) przedstawiającym przepływ prac:



Rys. 20. Diagram przedstawiający przepływ prac nad projektem „Clever Worm”

6.2. Struktura klas

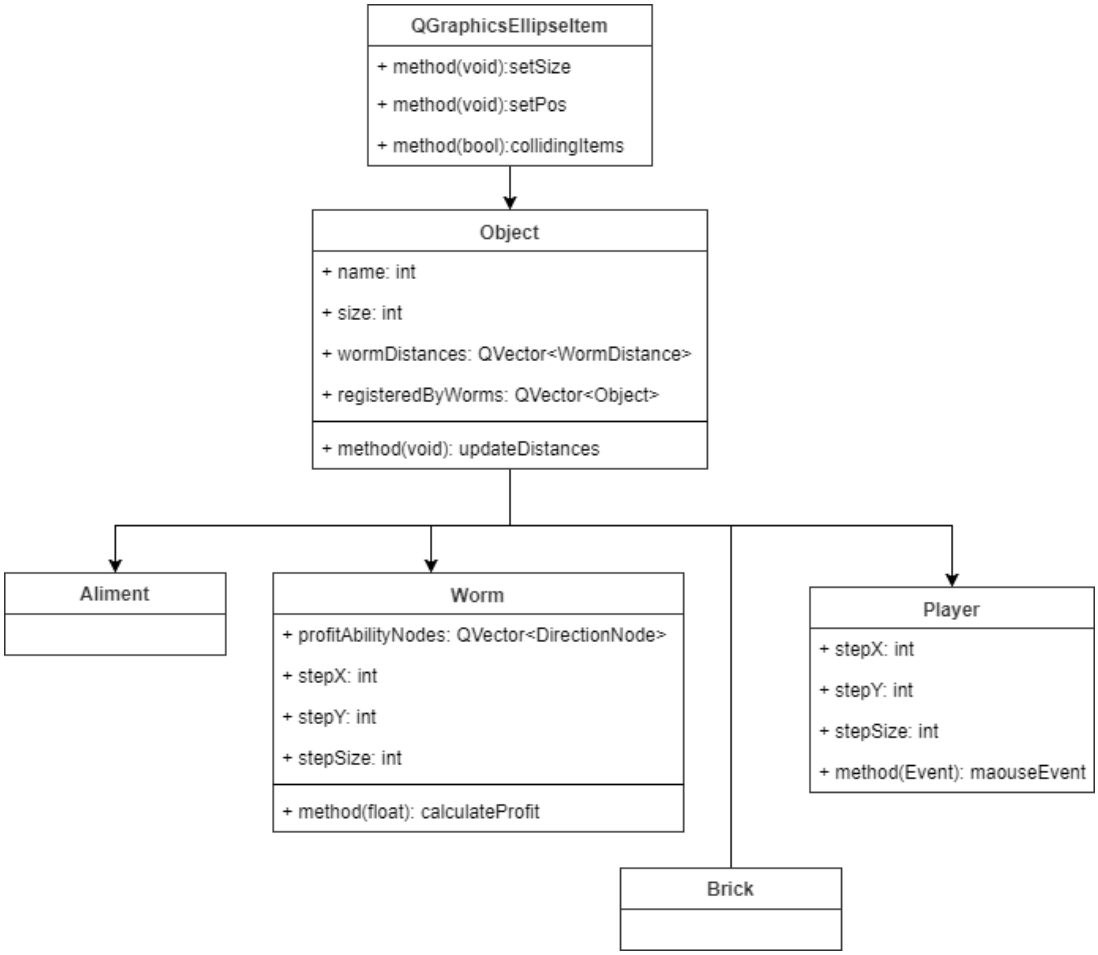
Struktura programu napisanego w języku obiektowym, do których bez wątpliwości należy język C++, polega w dużej mierze na dobrze skonstruowanej strukturze oraz precyzyjnie przemyślanych interakcjach, które mogą zachodzić między klasami. Ze względu na dużą ilość elementów tworzących grę postanowiono wzorować się na wzorcu projektowym „Budowniczy" (ang. *builder*), który zakłada budowanie obiektów zgodnie z zasadą „od szczegółu do ogółu". Obiekty mogą być rozmaitych postaci, a całość opiera się o jeden proces konstrukcyjny. Konkretny budowniczy decyduje o tym, jak dany obiekt jest tworzony, a szef zleca operacje budowniczemu. Na koniec wywoływane są wszystkie metody budowniczego i powstaje produkt końcowy[16]. W grze komputerowej „Clever Worm" za szefa odpowiada klasa „*Game*", budowniczym jest klasa „*GameWindow*". Dalsze interakcje względem wzorca zostały przedstawione na rysunku 21. Zaletami tego wzorca są możliwość zróżnicowania wewnętrznych struktur oraz duża skalowalność. Stosowanie go, zapobiega tworzeniu zduplikowanego kodu, ponieważ jeden kod reprezentuje wiele obiektów, a tworzenie obiektu ograniczone jest konkretną procedurą. Wadą natomiast jest duża ilość obiektów reprezentujących konkretne produkty.



Rys. 21. Przełożenie utworzonych klas na wzorzec projektowy „Budowniczy"

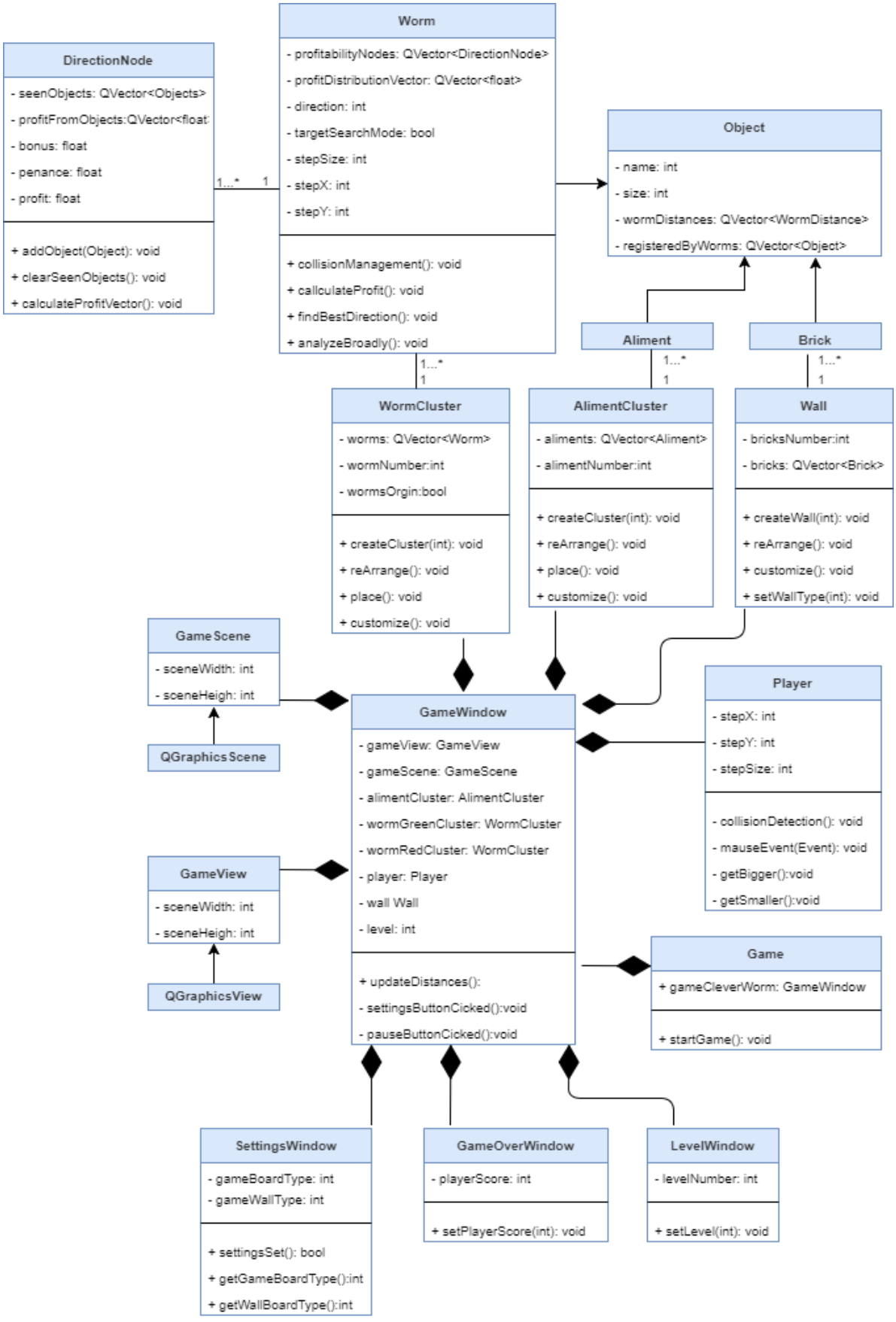
Innym, poza strukturą, równie ważnym elementem programowania obiektowego jest dziedziczenie. Pozwala ono na tworzenie spójnej, podatnej na modyfikacje hierarchii klas. Wykorzystywane jest wtedy, gdy jedna klasa współdzieli obiekty lub zachowania z inną klasą. Najlepiej gdy klasa pochodna jest szczególnym rodzajem klasy bazowej. Ponieważ w grze komputerowej „Clever Worm" wszystkie obiekty na scenie są typu „*QGraphicsItem*", to ta klasa uznana została za podstawową. Ze względu na fakt, że jest to klasa wbudowana w interfejs „*QGraphicsView"* i wykluczona jest jej modyfikacja, konieczne stało się stworzenie dodatkowej klasy, która reprezentowałaby nasze obiekty typu „*QGraphicsItem"*. Za klasę bazową, więc uznano abstrakcyjną klasę „*Object"*. Abstrakcyjność spowodowała, że

zabronione stało się utworzenie obiektu typu „*Object”*, a jedynie dziedziczenie po nim, co było w pełni zgodne z przyjętymi założeniami funkcjonalnymi. Rysunek 22 przedstawia wykorzystany mechanizm dziedziczenia, z wyszczególnieniem ważniejszych składowych i funkcji klas.



Rys. 22. Mechanizm dziedziczenia w grze „Clever Worm”

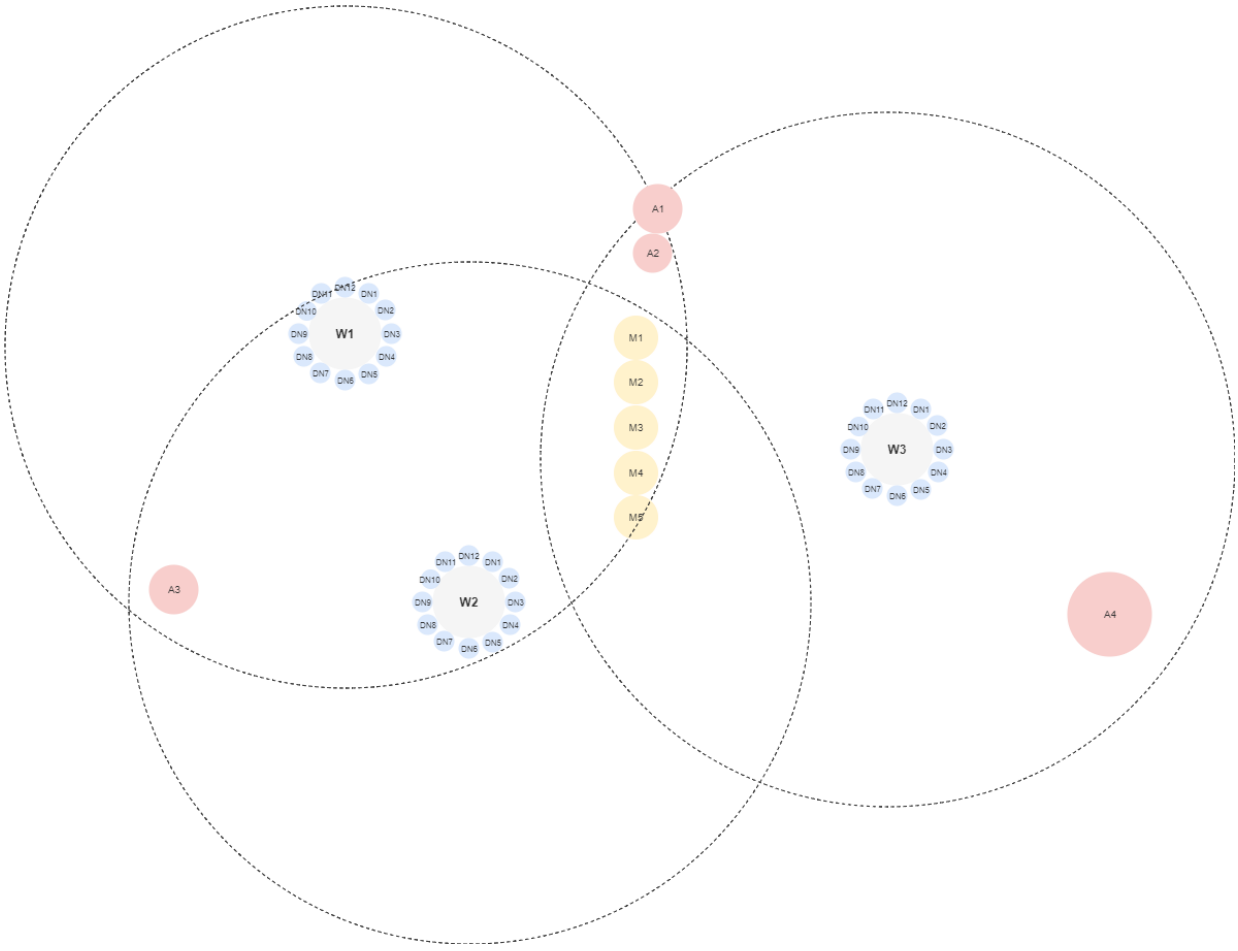
Powyższe diagramy reprezentują tylko pewne aspekty powstałej struktury. Do zilustrowania wszystkich klas i zależności między nimi wykorzystany został diagram klas zaprezentowany na rysunku 23. Przedstawia on każdy typ obiektowy (klasę) w programie, w odróżnieniu od diagramu obiektów, który pokazuje instancje obiektów i ich zależności w danym momencie. Diagram został stworzony w języku UML (*ang. Unified Modeling Language*). Za jego pomocą w sposób graficzny zaprezentowana została część szczegółów implementacyjnych.



Rys.23. Diagram klas dla projektu „Clever Worm”

6.3. Grafowy asocjacyjny model danych

Struktura grafowa służy do przedstawienia danych i relacji między nimi. Aby stworzyć graf, potrzebne są rekordy, które należy następnie odpowiednio przekształcić. Struktura grafowa, utworzona w grze „Clever Worm" jest o tyle nietypowa, że opiera się na dynamicznych, zmiennych w czasie obiektach. W tym rozdziale, aby przedstawić jej własności statyczne, omawiać ją będę uwzględniając tylko jedną chwilę czasową. Standardowo tworzenie grafu AGDS czy AANG, odbywa się za pomocą przekształcenia danych tabelarycznych w grafowe (np. metoda ASSORT [18]). W tym przypadku, dane są o tyle specyficzne, że tworzenie tabel mogłoby okazać się bardzo kosztowne, dlatego też dane gry zapisywane są bezpośrednio w grafowych strukturach asocjacyjnych. Wielkość struktury jest ściśle związana z ilością obiektów, stąd dla uproszczenia rozważań przedstawiony zostanie przykład dla: 3 obiektów typu „Worm", 4 obiektów typu „Pożywienie" oraz muru o długości 5 cegieł.

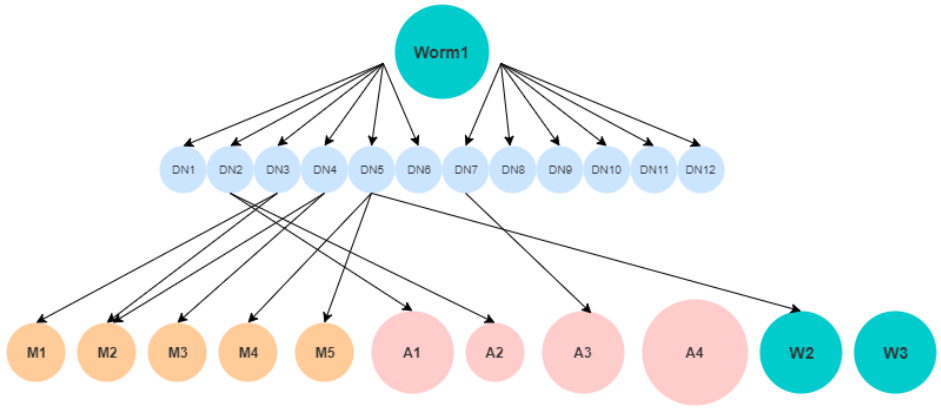


Rys.24. Obiekty, które przedstawione zostaną w asocjacyjnej grafowej strukturze

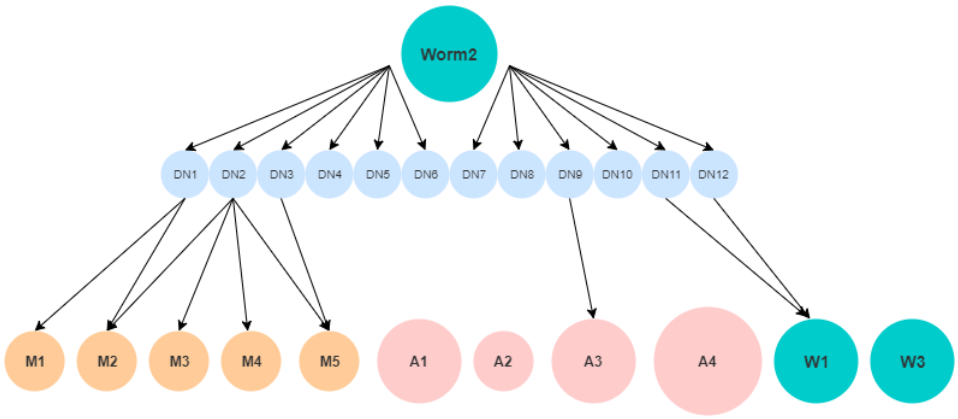
Na rysunku 24 zobrazowane zostało przykładowe ułożenie obiektów w grze *„Clever Worm"* dla określonej chwili czasowej. Linia przerywana symbolizuje zakres analizowanego obszaru, a małe kółka *„DN.."* reprezentują możliwe do obrania kierunki.

Graf asocjacyjny zbudowany jest z węzłów reprezentowanych przez wszystkie obiekty na scenie (gracz, zawodnicy, mur, pożywienie). Połączenia węzłów, czyli krawędzie

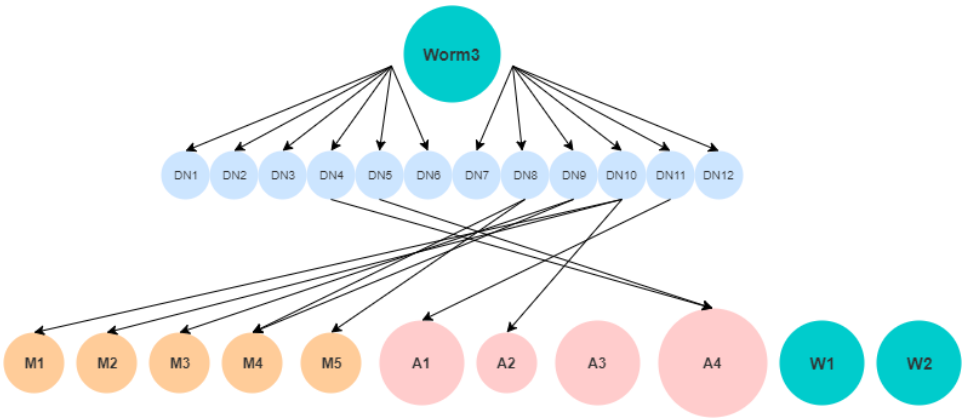
odzwierciedlają fizyczne połączenie obiektów, tzn. znajdowanie się w obszarze wzajemnej widoczności. Dla powyższego przykładu danych przedstawiono najpierw fragmenty grafu reprezentujące połączenia dla poszczególnych obiektów typu „Worm" (rysunek 25, 26, 27), a następnie wszystkie połącznia asocjacyjne między zaprezentowanymi obiektami.



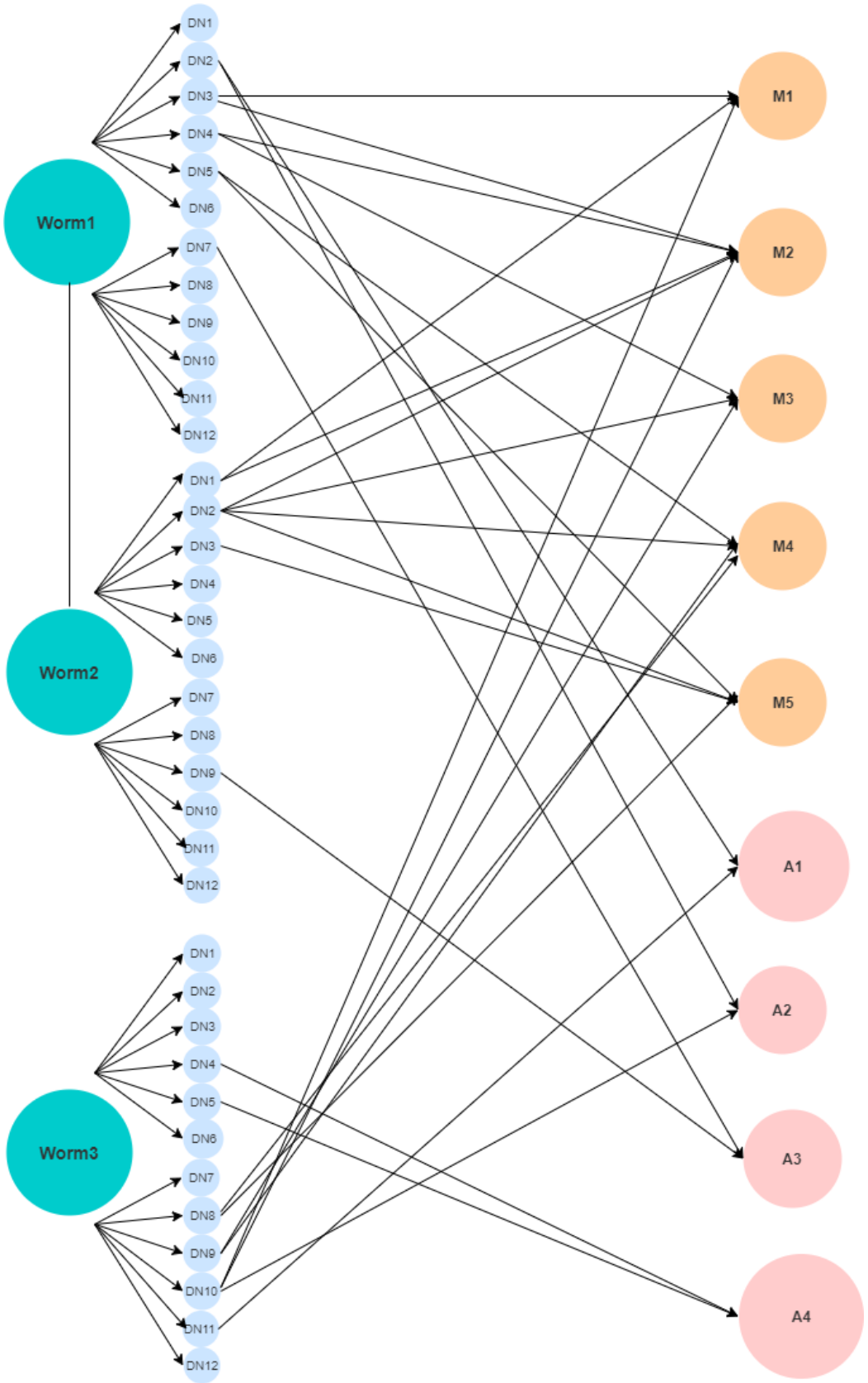
Rys. 25. Fragment grafu przedstawiający połączenia obiektu typu „Worm1" z innymi obiektami



Rys. 26. Fragment grafu przedstawiający połączenia obiektu typu „Worm2" z innymi obiektami



Rys. 27. Fragment grafu przedstawiający połączenia obiektu typu „Worm3" z innymi obiektami



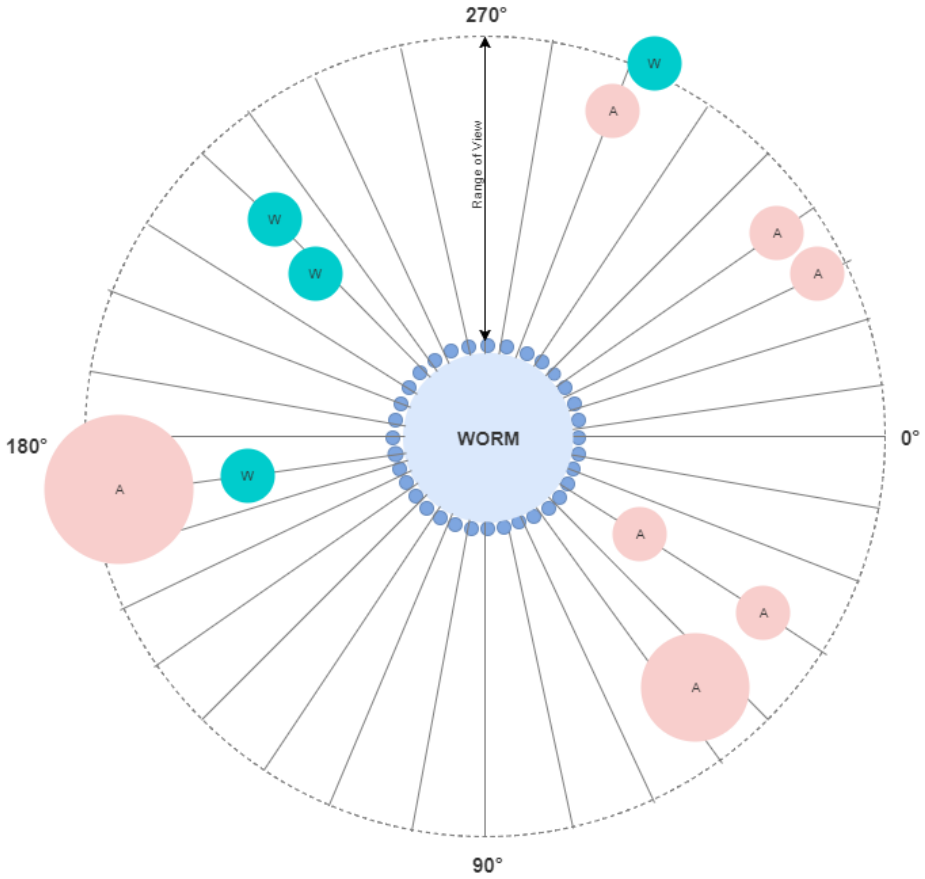
Rys. 28. Graf asocjacyjny przedstawiający obiekty z rysunku 24

7. Implementacja

7.1. Mechanizmy skojarzeniowe

7.1.1. Wykrywanie obiektów

Wszystkie obiekty widoczne na planszy są częścią grafu asocjacyjnego, dokładniej stanowią jego węzły. Każdy gracz automatyczny reprezentowany jest przez węzeł typu „Worm" oraz 36 dodatkowych węzłów „DN" (ang. *direction node*) przedstawiających możliwe do obrania kierunki. Każdy z węzłów DN przechowuje wektor wskaźników do widzianych obiektów z określonego obszaru. Obszar ten zdefiniowany jest jako okrąg o określonym wcześniej promieniu (ang. *range of view*).



Rys. 29. Obszar widoczności wraz z obiektami, które się w nim znajdują

Rejestrowanie obiektów przedstawionych na rysunku 29 można przedstawić za pomocą algorytmu:

Dla każdego węzła DN równego od i = 0 do i < 36:

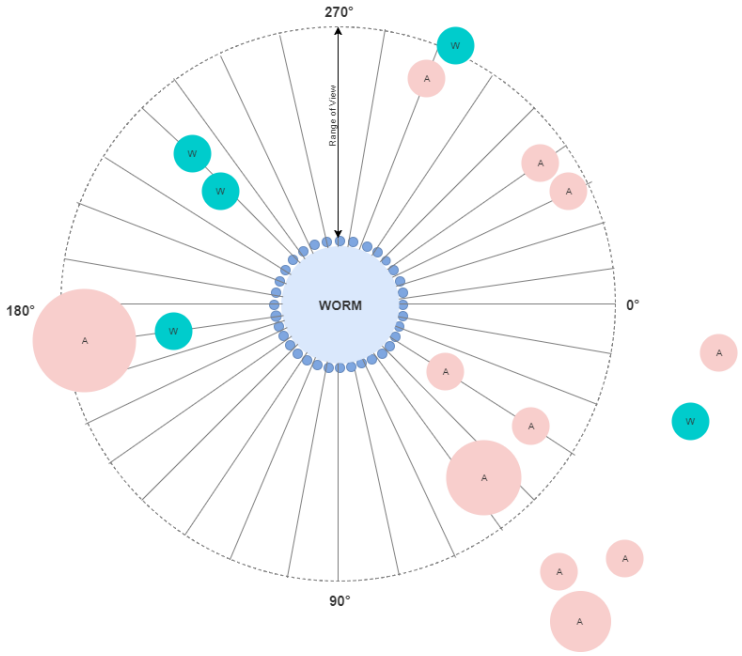
1) Sprawdź, co piąty punkt na planszy leżący wzdłuż wektora kierunku nie dalszy niż ustalony promień (wielkość pożywienia jest nie mniejsza niż 5, wyszukiwanie co 5 punktów jest elementem optymalizacji).

2) Jeśli w danym punkcie znajduje się obiekt, ustal, czy jest inny niż wcześniej już zarejestrowany.

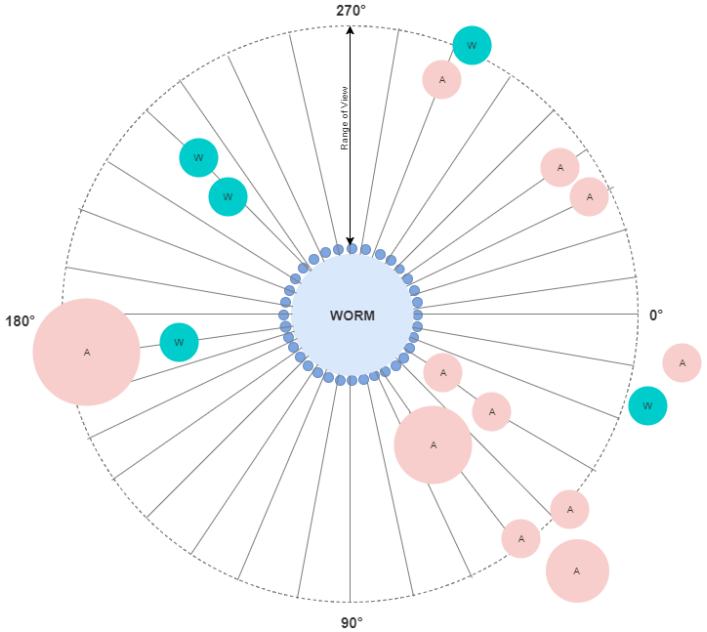
3) Określ typ znalezionego obiektu i dodaj go do listy wskaźników obiektów widzianych.

4) Zakończ przeszukiwanie i rozpocznij analizę.

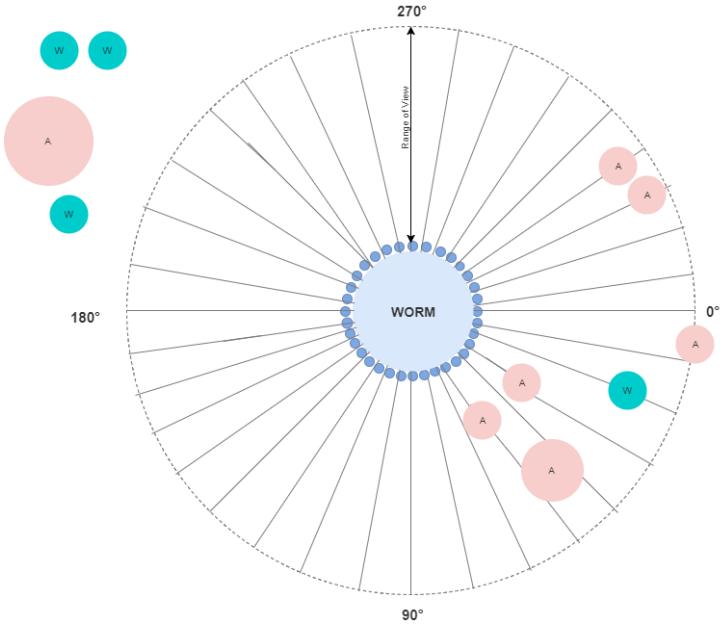
Co ustaloną chwilę czasową pozycja automatycznego zawodnika zmienia się, a wraz z nim przemieszcza się obszar jego widoczności. Dodanie nowych obiektów, które mieszczą się w zmienionym obszarze odbywa się również zgodnie z przedstawionym algorytmem. Na rysunkach 30, 31, 32 został przedstawiony obszar widoczności wraz z zarejestrowanymi obiektami.



Rys. 30. Obszar widoczności wraz z otoczeniem w chwili t_0



Rys. 31. Obszar widoczności wraz z otoczeniem w chwili t_1



Rys. 32. Obszar widoczności wraz z otoczeniem w chwili t_2

W chwili t_0 widzimy, że najkorzystniejszy kierunek to 45°. Kierunek ten zostaje wybrany i obiekt *Worm* zaczyna się w nim poruszać. Obszar przesuwa się, nowe obiekty zostają zauważone i dodane do wektorów kierunku zgodnie z przedstawionym wyżej algorytmem. W chwili t_1 mimo zmiany położenia, atrakcyjność kierunku jest nadal najwyższa, więc zostaje on utrzymany. W tej chwili ważny jest również fakt, że pomimo zmiany położenia obszaru widoczności, obiekty pomiędzy 180 a 270 stopniem są nadal obecne w wektorze kierunku, dzięki czemu informacja, którą ze sobą niosą nie jest utracona. Dopiero w chwili t_2 upatrzony cel zostaje spożyty, wektory kierunków czyszczą się, a następnie uzupełniają nowymi danymi.

7.1.2. Analiza otoczenia

Zebrane w węzłach kierunku dane o zauważonych obiektach stanowią bazę dla algorytmu. Zaraz po zakończeniu przeszukiwania, uruchamiane są funkcje kalkulujące profit. Najważniejszą i jednocześnie zmienną informacją, na podstawie której liczona jest atrakcyjność kierunku, jest odległość jaka dzieli badane obiekty. Każdy z obiektów na planszy posiada wektor wskaźników do obiektów, przez które został zarejestrowany (ang. *registered by worms*). Wskaźniki te umożliwiają obliczenie odległości, a następnie udostępnienie jej. Co pewien ustalony odstęp czasu, wektor odległości zostaje uaktualniony, tak że do obliczeń pobierana jest najnowsza wartość. Dzięki takiemu rozwiązaniu odległość jest swego rodzaju bodźcem, który dociera bezpośrednio do obiektu „*Worm*", a nie jest generowana przez przypisane do niego węzły kierunków. Metoda ta staje się bardzo korzysta, w szczególności, gdy dany obiekt został zauważony przez kilka różnych węzłów, a obliczanie odległości odbywa się tylko raz. Każdy obiekt zarejestrowany przez węzeł kierunku generuje profit. Jego wartość może być

dodatnia lub ujemna, w zależności od typu obiektu w stosunku do obiektu macierzystego. Wzór na profit dla obiektów wewnątrz obszaru widoczności można zapisać za pomocą:

$$p = \frac{R - d}{R} \cdot S \qquad (7.1.2.1)$$

gdzie *p* – profit, *R* – promień obszaru widoczności (ang. *range of view*),
d – odległość (ang. *distance*) i *S* – wielkość obiektu (ang. *size*).

Gdy obiekt znajduje się poza obszarem widoczności, ale został zarejestrowany i docierają do nas informacje o jego położeniu, profit wyliczany jest za pomocą:

$$p = \frac{1}{d} \cdot S \qquad (7.1.2.2)$$

Jeśli profit jest dodatni, obiekt po kontakcie z nim zyskuje swego rodzaju bonus. Nagroda zależna jest od wielkości profitu i wiąże się ze zwiększeniem masy oraz uzyskaniem dodatkowych punktów dla drużyny. Gdy profit jest ujemny, skutki są analogiczne, lecz negatywne, obiekt maleje i jego drużyna traci punkty. Co więcej w przypadku kontaktu z groźnym (czytaj znacznie większym przeciwnikiem) możliwa jest również utrata życia. Profit zaliczany jest do negatywnych, gdy:

- obiekt jest przeszkodą statyczną – mur,
- obiekt jest przeszkodą dynamiczną, należącą do własnej drużyny.

Profit zaliczamy do pozytywnych, gdy:

- obiekt jest pożywieniem,
- obiekt jest mniejszą przeszkodą dynamiczną, należącą do przeciwnej drużyny.

Standardowo profit dla węzła kierunku jest wypadkową wszystkich profitów zarejestrowanych od zauważonych obiektów. Jeśli wypadkowa jest dodatnia, mówimy o nagrodzie, gdy jest ujemna, z danym kierunkiem powiązana jest kara. Dla niektórych przypadków wprowadzono procedurę, pozwalającą na nałożenie dodatkowej dużej kary. Sytuacja ta ma miejsce, gdy najbliższy obiekt jest typu „Worm". Zabezpieczenie to ma powodować całkowite zablokowanie takiego kierunku.

7.1.3. Podjęcie i utrzymanie decyzji

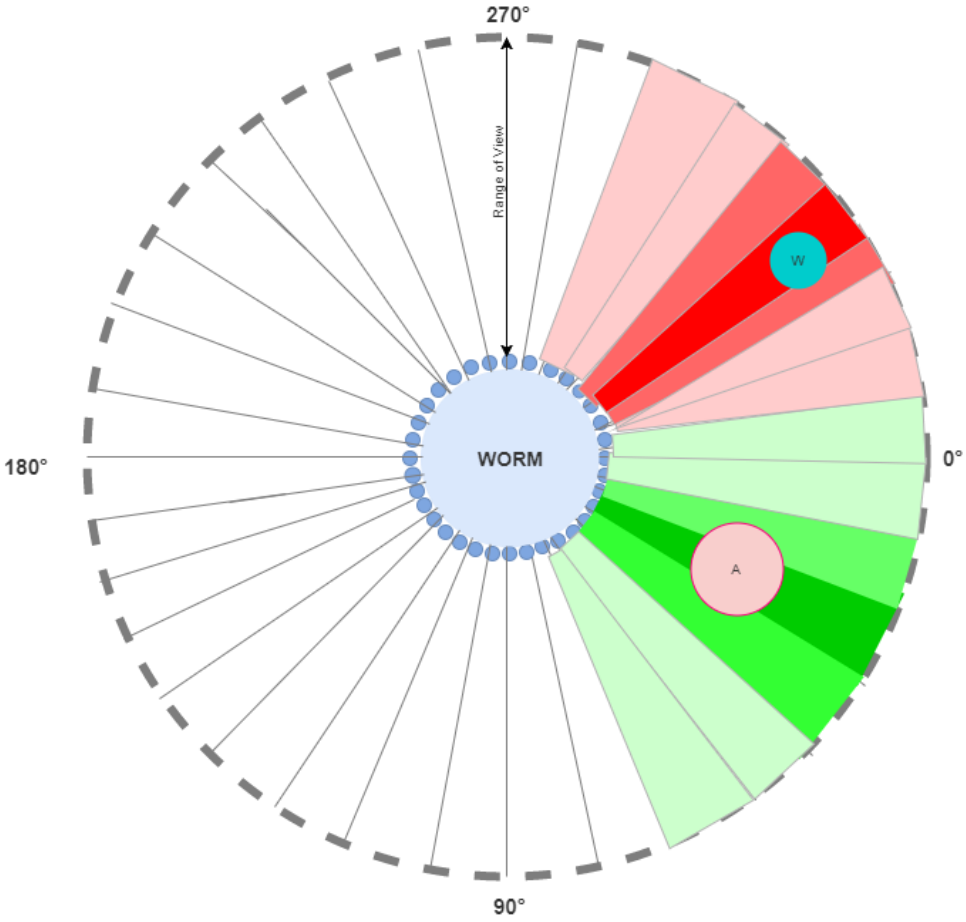
Obiekt analizuje przestrzeń w 36 kierunkach, które są od siebie nachylone pod kątem 10°. Dla każdego z tych kierunków utworzona zostaje wypadkowa nagrody i kary, na podstawie, której podjęta zostaje decyzja. Ostateczną wartość mówiącą o atrakcyjności danego kierunku możemy wyrazić za pomocą wzoru:

$$A = 0{,}7\,p + 0{,}075(p1 + p2) + 0{,}0375(p3 + p4 + p5 + p6) \qquad (7.1.2.3)$$

gdzie *A* – atrakcyjność kierunku, *p* – profit z macierzystego węzła, *p1*, *p2* – profit z przyległych węzłów, *p3*, *p4*, *p5*, *p6* – profit z kolejnych najbliższych węzłów.

Dzięki takiemu rozłożeniu, kierunek nałożony dużą karą (z zarejestrowanym wrogim obiektem dynamicznym) ma wpływ na otaczające go węzły, co umożliwia zablokowanie

niebezpiecznego kierunku i najbliższego otaczającego go obszaru. Rysunek 33 prezentuje przykładowe oddziaływanie kierunku z dużą karą i kierunku z dużą nagrodą.



Rys. 33. Wpływ profitu węzła na sąsiadów

Celem automatycznego zawodnika jest uzyskanie możliwe największej liczby punktów, a co za tym idzie zdobycie pożywienia lub pozbycie się zawodnika drużyny przeciwnej. Algorytm opisujący podejmowanie decyzji:

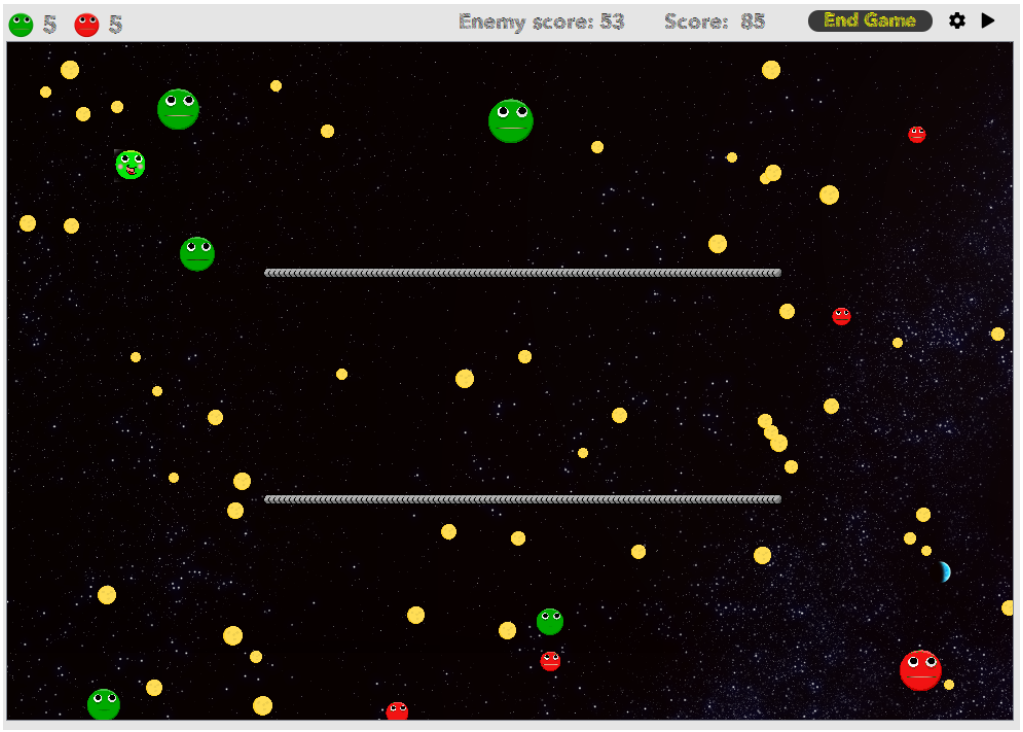
1) Przeszukaj otoczenie w trybie „szukaj celu” – promień widzianego obszaru jest dwukrotnie większy.
2) Zapamiętaj kierunek z obranym celem.
3) Przeszukuj otoczenie w normalnym zakresie w celu wykrycia ewentualnych zagrożeń.
4) Doładuj wartość profitu dla wybranego kierunku poprzez obliczenia po odebranych bodźcach.
5) Wybierz kierunek zapewniający największą nagrodę, jeśli podczas wykonywania punktu 3 nie pojawiło się zagrożenie, kierunek jest zgodny z kierunkiem z punktu 2.
6) Potarzaj kroki od 3 do 5 aż dojdzie do kolizji z wybranym obiektem.
7) Wyczyść węzły kierunków i przejdź do punktu 1.

7.2. Elementy graficzne i dźwiękowe

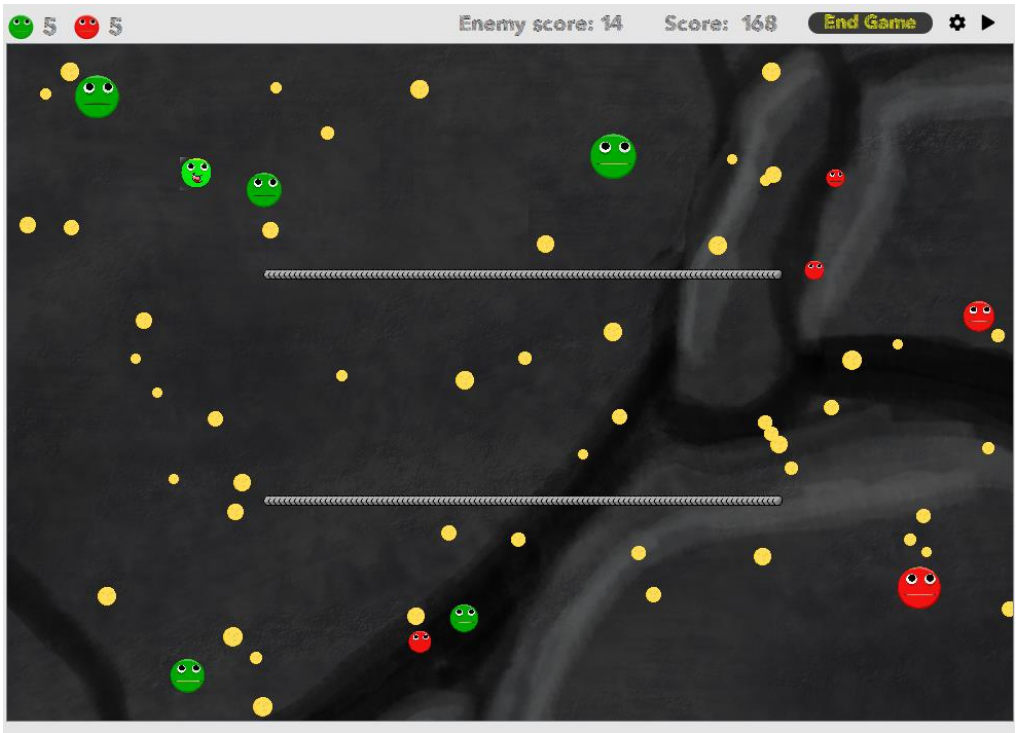
W grze „Clever Worm” obrazy i muzyka są elementem drugoplanowym. Tworzą one harmonijny układ i w nieznacznym stopniu zwiększają atrakcyjność gry. Grafika sama w sobie oddziałuję na jeden z ważniejszych zmysłów – wzrok, stąd nie może zostać całkowicie pominięta. Gra komputerowa podobnie jak jedzenie, bardzo często oceniana jest po wyglądzie, dlatego postanowiłam, że najlepszym określeniem grafiki w grze „Clever Worm” będzie przymiotnik „estetyczna”. W grze dominują kolory takie jak: biały, czarny, żółty i niebieski. Zawodnicy, pożywnie i gracz posiadają intensywniejsze barwy, aby przykuwać uwagę. Drużyna gracza jest zielona, ponieważ ten kolor ma relaksujący i odstresowujący wpływ, natomiast jej rywale, są koloru czerwonego, który utożsamia się z zagrożeniem, ostrzeżeniem, ruchem i działaniem. Elementy gry to proste dwuwymiarowe figury geometryczne, koła i wielokąty.

Obiekty z planszy jak i same motywy utworzone zostały w programie GIMP (*ang. GNU Image Manipulation Program*), który umożliwia edycję grafiki rastrowej. Obraz w grafice rastrowej reprezentowany jest przez prostokątną siatkę leżących obok siebie punktów zwanych pikselami. Za bazę części elementów gry służyły ogólnodostępne symbole ze strony *http://www.reinerstilesets.de/*. Edytor GIMP pozwolił na ich odpowiednie przekształcenie i dopasowanie do panującego stylu. Następnie, obrazy te zostały wczytane jako obiekty klasy *QPixmap*. Klasa ta pobiera bitmapę, dostosowuje jej kształt, a później prezentuje ją na ekranie. Do samego rysowania elementów GUI wykorzystany został również *QPainter*, który umożliwiał optymalne aktualizowanie obrazu, między innymi jego zwiększanie podczas pozyskiwania pożywienia. Dodatkowo krawędzie elementów poruszających się na planszy, zostały lekko wygładzone za pomocą klasy *QGraphicsBlurEffec*. Rysunki 34, 35 i 36 przedstawiają dostępne szaty graficzne gry „Clever Worm”.

Rys. 34. Plansza graficzna reprezentująca dżunglę



Rys. 35. Plansza graficzna reprezentująca kosmos



Rys. 36. Plansza graficzna reprezentująca podziemia

Oprawa muzyczna gry ograniczała się do wprowadzenia klasycznych dźwięków towarzyszących podstawowym akcjom gracza. Muzyka została pozyskana z darmowych źródeł, co nie zmienia faktu, że umieszczenie wymagało pewnego nakładu pracy. Pierwszy krok stanowiło wybranie momentu, w którym dźwięk miał się pojawić, następnie należało znaleźć odpowiedni sampel i dostosować go do potrzeb (przyciąć, pozbyć się ewentualnych

szumów, ustawić głośność). Elementy dźwiękowe typu „feedback" zostały wprowadzone do gry za pomocą klasy *QSoundEffect* znajdującej się w pakiecie multimedia. Klasa ta umożliwiała wprowadzenie nieskompresowanych plików dźwiękowych (np. *wav*), które reprezentowały kolizję gracza z innymi elementami na planszy. Do takich dźwięków należało zjedzenie pożywienia, zjedzenie pożywienia zwiększającego prędkość, dźwięk kolizji z graczem ze swojej drużyny i sygnał końca gry. Inny rodzaj dźwięków, cykliczny występujący w tle, został wprowadzony za pomocą klasy *QMediaPlayer*. Ostatnim elementem prac nad dźwiękiem było przetestowanie efektu końcowego.

7.3. Testy

Testowanie gier komputerowych, podobnie jak klasyczne testowanie oprogramowania ma na celu weryfikację stworzonego software'u, umożliwia też utrzymanie go zgodnie z założeniami jakościowymi. Zakres testów gry „Clever Worm" można podzielić następująco:

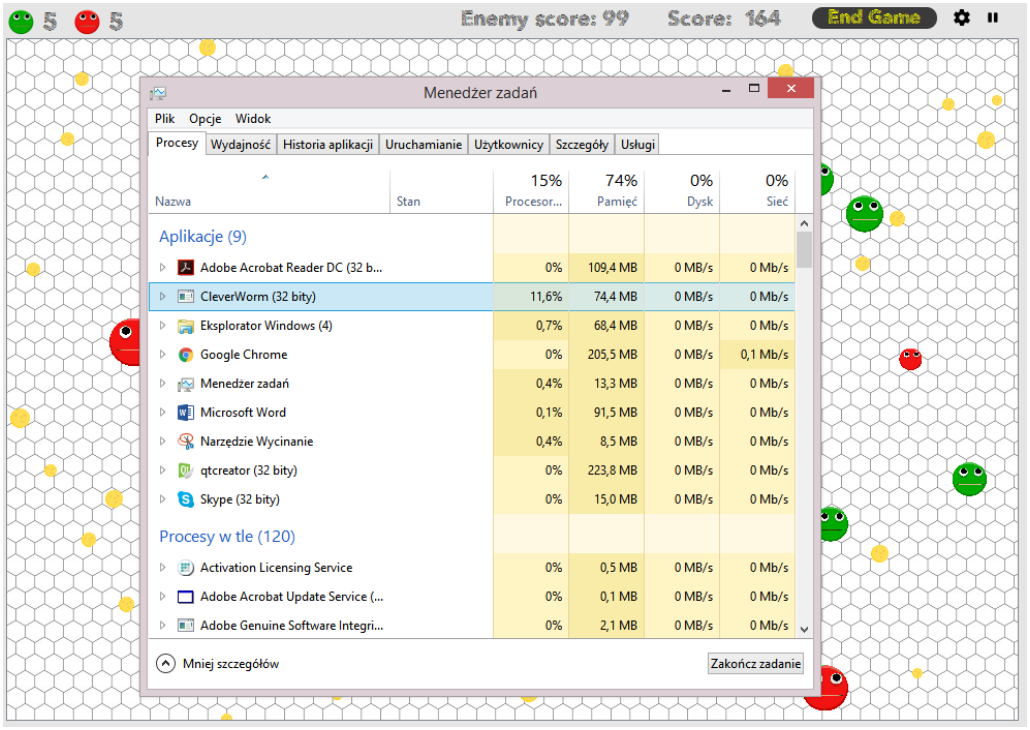
- testy funkcjonalne,
- testy stabilności,
- testy wydajnościowe,
- testy struktury skojarzeniowej.

Testy funkcjonalne wykonywane były zgodnie z metodyką agile. Każdorazowo po implementacji nowej funkcjonalności weryfikowano działanie wszystkich mechanizmów gry, testowano jej atrakcyjność i stabilność. Następnie przy kolejnej zmianie pod uwagę brana była kolejna funkcjonalność i ewentualne wcześniej wprowadzone poprawki. Po zaimplementowaniu wszystkich funkcjonalności na wadze przybrały testy kosmetyczne oraz balansowanie rozrywki. Ostateczna wersja gry poddana została testom przeprowadzonym przez kilkuosobową grupę, a uwagi wówczas zgromadzone zostały w miarę możliwości zaaplikowane.

Testy stabilnościowe, były również przeprowadzane cyklicznie, jednak nie tak często jak testy funkcjonalne. Stabilność sprawdzana była bowiem tylko przy większej zmianie. Testowano różne przypadki użycia oraz reakcję aplikacji na nie. Duża część testów wykonywana była za pomocą wbudowanego debuggera, który pozwalał kontrolować zalokowaną pamięć czy podglądać wartości używanych zmiennych. Głównym celem testów stabilnościowych było znalezienie przypadków, po których gra się zacina i następnie wyeliminowanie ich.

Kolejny rodzaj testów, czyli testy wydajnościowe, stał się kluczowy w ostatnich wersjach aplikacji. Sprawdzano ogólne zużycie pamięci oraz wpływ na nie elementów muzycznych i graficznych. Zużycie to zostało sprawdzone za pomocą Menagera Zadań i jak widać na rysunku 34 wynosi ono 74 MB. Wynik ten nieznacznie zwiększał się wraz z osiągnięciem kolejnego poziomu. Poza zajętością pamięci, ważny był również odbiór gry pod kątem wydajnościowym. Automatyczni zawodnicy powinni poruszać się z góry określoną

prędkością, a gracz zdalny powinien zachowywać swoją zwinność, nawet przy zwiększonej liczbie obiektów na planszy.



Rys. 37. Pamięć zajmowana przez proces gry „Clever Worm”

Ostatni rodzaj testów polegał na testowaniu struktury skojarzeniowej. Jego celem była weryfikacja zaimplementowanych algorytmów oraz ocena ich skuteczności. Pierwszy etap rozpoczął się dla bardzo podstawowej wersji gry i polegał na analizie ruchów wbudowanego zawodnika. Pożywienie jak i gracz umieszczane były w sprecyzowanej lokalizacji a za pomocą obliczeń algebraicznych, przeprowadzana była weryfikacja poprawności wybranych kierunków. Obliczenia algebraiczne sprowadzały się do wykorzystania wzorów *7.1.2.1* i *7.1.2.2*. Testy te były powtarzane po każdej większej zmianie algorytmu. W znacznej większości przypadków wynik obliczeń potwierdzał zachowanie automatycznego zawodnika. Dodatkowo weryfikowano reakcję gracza na przeszkody statyczne i dynamiczne. W przypadku przeszkód statycznych, wyniki testów był również pozytywne. Dla testów z obiektami dynamicznymi, około 10% przypadków nie zmieniło kierunku na czas. Jak się okazało, czas uaktualniania położenia, długość kroku oraz ilość otaczających obiektów branych do analizy miały wpływ na szybkość reakcji na niebezpieczeństwo. Za optymalne wartości uznano update, co 100 milisekund, długość kroku 5 oraz wpływ otaczających 6 wartości węzłów kierunków.

8. Wnioski

Tematem pracy był system inteligentnego kontekstowego wnioskowania i sterowania grą komputerową, wykorzystujący asocjacyjny grafowy model danych i metody inteligencji obliczeniowej. W ramach pracy utworzono grę komputerową o nazwie „*Clever Worm*", zaimplementowano grafowe struktury skojarzeniowe oraz algorytmy pozwalające na kontekstową analizę otoczenia. Stworzono automatycznych zawodników poruszających się w sposób kreatywny, nakierowanych na zdobycie nagrody i uniknięcie kary oraz gracza komputerowego, który rywalizuje z nimi w walce o pożywienie.

Pierwszym elementem części teoretycznej wykonywanego projektu było zapoznanie się z tematyką tworzenia gier komputerowych. Początkowo przytoczono definicję samej gry komputerowej oraz określono cechy dla niej charakterystyczne. Następnie przedstawione zostały początki gier ze szczególnym uwzględnieniem poprzedników tworzonej gry, czyli wszystkich gier typu „*Snake*". Opisany został proces tworzenia gier oraz jego zaawansowanie, które jest proporcjonalne do wielkości.

W następnym rozdziale przedstawiono czytelnikowi teorię dotyczącą grafowych asocjacyjnych struktur danych. Omówione zostało pojęcie sztucznej inteligencji. Przedstawiono pojęcie danych i informacji oraz zależności między tworzeniem się informacji i kształtowaniem wiedzy. Opracowanie to okazało się niezwykle pomocne w późniejszej implementacji algorytmów kontekstowych. Dodatkowo przytoczone zostało pojęcie asocjacji i założenia określające mechanizmy skojarzeniowe. Pod koniec rozdziału wyszczególniono różne rodzaje połączeń skojarzeniowych oraz zaprezentowano wykorzystujące je dwie struktury grafowe AGDS oraz AANG.

Rozdział 4 opisuje wykorzystane środowisko oraz język programowania. Przedstawiono biblioteki takie jak *QGraphicsView*, które uogólniając pozwoliły na implementację gry. W tym rozdziale opisane jest również wykorzystanie poszczególnych ich klas w tworzeniu elementów gry.

Rozdział 5 jest pierwszym rozdziałem opisującym część praktyczną wykonanego projektu. Przedstawia on elementy gry oraz jej mechanikę. Zaprezentowane zostają w nim też możliwe akcje gracza oraz utworzona szata graficzna.

Kolejny rozdział jest opisem tworzenia projektu. Przedstawiony został cykl wszystkich prac oraz architektura systemu. Opisane zostały mechanizmy oraz struktura dziedziczenia. Rozdział ten jest o tyle istotny, że pokazuje skomplikowanie projektu oraz jego strukturę.

Rozdział 7 jest kluczowy. Przedstawiono w nim praktyczne zastosowanie struktur opisanych w rozdziale 3. Zaprezentowano metodę tworzenia asocjacyjnej struktury grafowej oraz jej faktyczne zastosowanie podczas analizy otoczenia. Opisany został algorytm wykrywania nowych obiektów oraz pozytywy wynikające z chwilowego przechowywania informacji o ich położeniu. Zaprezentowana została również metoda pozwalająca na

uogólnienie i analizę kontekstową, jaką jest obliczenie profitu na podstawie ustalonej liczby otaczających węzłów kierunku.

W kolejnym rozdziale przedstawione zostały różne techniki testowania oraz ich wpływ na zmiany w aplikacji.

„*Clever Worm*" jest prostą grą zręcznościową. Jej atrakcyjność oparta jest o zaimplementowane algorytmy skojarzeniowe oraz klarowną fabułę. W chwili pisania pracy (sierpień 2017), czyli po pół roku pracy nad projektem można powiedzieć, że gra jest nadal udoskonalana i dopracowywane są jej szczegóły. Wynika to z faktu, że tworzenie gier jest czynnością niezwykle wciągającą, a pokusa ciągłego udoskonalania jest bardzo duża. Potrzeba perfekcjonizmu góruje czasem nad zdrowym rozsądkiem, co pozwala mi stwierdzić, że twarde wymagania powinny zostać zdefiniowane nieco wcześniej niż odbyło się to w moim przypadku. Chwilowa postać gry jest dla mnie zadowalająca, jednakże zdaje sobie sprawę z szerokich możliwości jej rozwinięcia. Jednym z takich przykładów mogłaby być zmiana jednego z kluczowych założeń mówiących, że celem gracza automatycznego nie jest tylko zdobycie pożywienia, ale również unikanie zagrożeń. Wówczas prawdopodobnie możliwy byłby do zrealizowania efekt uciekania jeszcze w większym stopniu niż obecnie, co moim zdaniem zwiększyłoby atrakcyjność gry. Dodatkowo oczywiście możliwa byłaby edycja grafiki czy elementów muzycznych.

9. Bibliografia

[1] *Słownik Języka Polskiego*,[online] ostatnia wizyta: 29.09.2017, dostępny pod *https://sjp.pwn.pl/sjp/gra-komputerowa;2462665.html*,
[2] *Wikipedia, Wolna Encyklopedia*, [online] ostatnia wizyta: 29.09.2017 dostepny pod *https://pl.wikipedia.org/wiki/Gra_komputerowa*,
[3] J. Saucier, *Atari Packaging Design Collection Documents the Art of Selling Video Game*, [online], aktualizacja: 17.12.2015, dostępny pod *http://www.museumofplay.org/blog/chegheads/2015/12/atari-packaging-design-collection-documents-the-art-of-selling-video-games/*
[4] J. Hawkins, S. Blakeslee, *Istota inteligencji*, 2005, OnePress, s. 13-27
[5] survey.pdf, s.10, [online] ostatnia wizyta: 29.09.2017 dostępny pod *http://www.nickbostrom.com/papers/survey.pdf*
[6] W. Duch*, Dokąd zmierza inteligencja obliczeniowa?*, Katedra Informatyki stosowanej Uniwersytetu Mikołaja Kopernika
[7] A. Horzyk, *Sztuczne systemy skojarzeniowe i asocjacyjna sztuczna inteligencja*, Warszawa 2013, Akademicka Oficyna Wydawnicza EXIT
[8] M. Grabowski, A. Zając, *Dane, informacja, wiedz – próba definicji*, „Zeszyty Naukowe Uniwersytetu Ekonomicznego w Krakowie", 2009, nr 798, s,.99-116
[9] *Słownik Języka Polskiego*,[online] ostatnia wizyta: 29.09.2017, dostępny pod *https://sjp.pwn.pl/slowniki/kojarzenie.html*
[10] *Qt Wiki About Qt* [online] ostatnia wizyta: 29.09.2017, dostępny pod *https://wiki.qt.io/About_Qt*
[11] *Qt Wiki Qt History* [online] ostatnia wizyta: 29.09.2017, dostępny pod *https://wiki.qt.io/Qt_History*
[12] *Qt Documentation* [online] ostatnia wizyta: 29.09.2017, dostępny pod *http://doc.qt.io/qt-5/graphicsview.html*
[13] M. Schoenhttps, *Why is C++ considered the best language for professional game development?*, [online], aktualizacjia: 23.06.2015, dostępne pod *https://www.quora.com/Why-is-C++-considered-the-best-language-for-professional-game-development*
[14] *SOLID – Dobre praktyki programowania*, [online], aktualizacja: 30.03.2016, dostępne pod *https://sii.pl/blog/solid-dobre-praktyki-programowania/*
[15] Robert C. Martin, *Czysty kod*, 2014, Helion
[16] *Wzorce projektowe* [online] ostatnia wizyta: 29.09.2017, dostępny pod *http://www.algorytm.org/wzorce-projektowe/budowniczy-builder.html*
[17] W. Wysota, L.Hass, *Game Programming Using Qt*, Pack Publishing 2016
[18] A. Horzyk, *Neurons Can Sort Data Efficiently*, In: Rutkowski L., Korytkowski M., Scherer R., Tadeusiewicz R., Zadeh L., Zurada J. (eds), Artificial Intelligence and Soft

Computing, Proc. of ICAISC 2017, Springer-Verlag, LNCS, Vol. 10245, pp. 64-74, 2017, DOI: 10.1007/978-3-319-59063-9_6