Autor projektu

Imię i nazwisko: Michał Kowalczyk
Numer albumu: 310762
E-mail: 01158759@pw.edu.pl

Rozwiązanie

Struktura kodu

Kod jest podzielony na 4 moduły, plik służące do testów jednostkowych, 2 pliki wygenerowane za pomocą Qt Designer i kod służący do startu aplikacji.

Moduły to:

- mark_class.py
- quantum_tic_tac_toe_class.py
- player_class.py
- quantum_tic_tac_toe_gui.py

Plik służące do testów jednostkowych to:

- test_classes_quantum_tic_tac_toe.py

Pliki wygenerowane za pomocą Qt Designer i przekonwertowane na pythona to:

- ui_quantum_tic_tac_toe.py
- ui_rules_window.py

A główny plik to:

- quantum_tic_tac_toe.py

Struktura rozwiązania

Struktura rozwiązania podzielona jest między 4 główne klasy:

- **Mark** – reprezentujący znak postawiony na planszy kwantowego kółka i krzyżyka
- **Quantum_Tic_Tac_Toe** – reprezentujące planszę gry
- **Player** – reprezentujące gracza, który gra w kwantowe kółko i krzyżyk
- **QuantumTicTacToeWindow** – okienko aplikacji

Mark ma prywatne 4 atrybuty:

- mark – string z „x” albo „o” oznaczający to jakim jest znakiem
- entanglement – lista z wartościami dwóch kwadratów, w których został postawiony
- move_number – numer ruchu, w którym został postawiony
- collapsed – prawda lub fałsz, oznaczające to, czy znak został zmierzony

Qunatum_Tic_Tac_Toe ma 4 prywatne atrybuty:

- squares – słownik z kluczami od 1 do 9 które reprezentują kolejne kwadraty planszy. Na starcie zawarte są listy [False], pierwsze pole listy reprezentuje to, czy dany kwadrat został zmierzony, a pozostałe reprezentują znaki, które zostały postawione na planszy
- last_placed_mark – zmienna klasy **Mark** ostatnio postawiony na planszy gry znak
- unresolved_cycle – prawda lub fałsz, prawda oznacza, że ostatnio położony znak zaczął cykl i trzeba się tym zająć
- paths – lista, na której odkładamy powstające ścieżki między znakami, ścieżki reprezentowane są przez zbiory. Na początku gry jest pusta, ale wraz z dodaniem kolejnych znaków się aktualizuje

Player ma prywatne 2 atrybuty:

- mark - string z „x” albo „o” oznaczający to jakim znakiem w tym momencie gra gracz
- score – int oznaczający wynik gracza

Sama gra odbywa się przez komunikację pomiędzy tymi trzema klasami - bez pomocy **QuantumTicTacToeWindow**, więc zostanie ono omówione później.

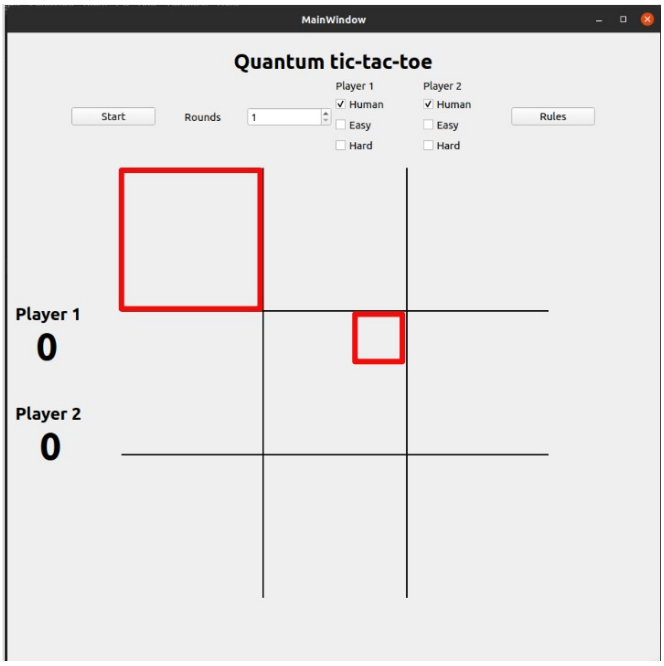
Najważniejsze metody **Quantum_Tic_Tac_Toe** to:

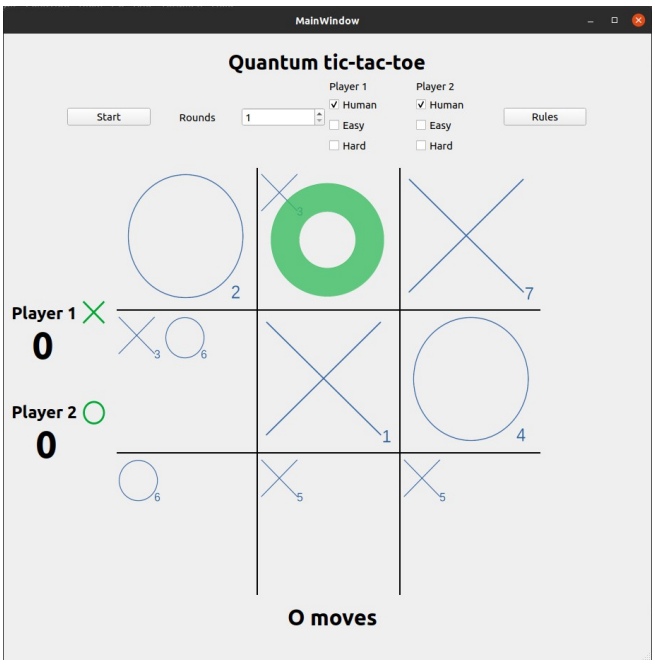
- add_entangled_mark(), które dodaje standardowy znak typu **Mark** na odpowiednie pola squares, uaktualnia last_placed_mark, paths, jeśli wystąpił cykl po dodaniu znaku zmienia wartość atrybutu unresolved cycle na True
- collapse_squares(), które jeśli unresolved cycle ma wartość True po podaniu startowego znaku i kwadratu, w którym chcemy, żeby się znajdował uaktualnia wszystkie klucze squares i symuluje działanie mierzenia cyklu
- one_round(), które restartuje klasę, przeprowadza za pomocą graczy **Player** jedną rundę gry i zwraca wynik
- both_cycle_resolution_options(), które, jeśli unresolved cycle ma wartość True, tworzy dwie głębokie kopie gry, symuluje na nich wybór dwóch różnych pól, w których można zmierzyć znak, który spowodował wystąpienie cyklu

Gracz **Player** ma

- collapse_choice() - po podaniu gry **Quantum_Tic_Tac_Toe** zwraca wybór gracza co do tego, w którym z dwóch kwadratów ma zostać zmierzony znak
- mark_choice() - po podaniu gry **Quantum_Tic_Tac_Toe** zwraca wybór gracza co do tego, gdzie powinien postawić znak

Ponadto każdy z graczy ma _mark_decision() i _collapse_decision(), które de facto decydują o tym w jaki sposób dana podklasa gracza decyduje, w tych sytuacjach. To rozróżnienie było konieczne ze względu na GUI, i pozwoliło łatwiej przenieść ruch graczy komputerowych niewspółpracujących z GUI na analogicznych współpracujących z GUI.
Plansza gry w interfejsie zawiera 9 przezroczystych przycisków **QPushButton** i 9 etykiet **QLabel,** które wyświetlają już zmierzone znaki i zaznaczenia, pod każdym przyciskiem jest poza tym 9 kolejnych etykiet, które wyświetlają kolejne postawione niezmierzone znaki.





Informacje techniczne

Wykorzystane moduły i narzędzia

Do stworzenia GUI wykorzystany został Qt Designer, więc jednym z podstawowych modułów było `PySide2`. Kolejnym wykorzystanym modułem było `datetime` do zaimplementowania qWait (funkcja została wzięta stąd: https://stackoverflow.com/questions/17960159/qwait-analogue-in-pyside). Poza tym do losowego wyboru działań graczy potrzebny był `random` i do stworzenia głębokiej kopii gry potrzebne było `copy`. Do wykonania ikon wykorzystane zostało LibreOffice Draw. Do testowania standardowo przydał się `pytest`.

Instrukcja instalacji

Niestety były problemy z wykorzystaniem pyinstallera, a sam program zajmował 0,5 GB, więc żeby włączyć program trzeba włączyć za pomocą pythona quantum_tic_tac_toe.py

Testy

Testy jednostkowe znajdują się w test_classes_quantum_tic_tac_toe.py. GUI niestety nie zostało przetestowane za pomocą testów jednostkowych. Testy jednostkowe pokazały, że komputerowy gracz o bardziej zaawansowanym AI prawdopodobnie zachowuje się tak jak powinien, ale w żadnym wypadku nie zaszkodziłoby przetestowanie jego zachowania przy paru innych sytuacjach na planszy. Testy wykazały też, że korzystając z publicznych funkcji nie ma możliwości popsucia gry bez zasygnalizowania tego błędem.

Wnioski i obserwacje

Programowanie obiektowe jest bardzo wygodne do realizacji takich większych programów i przy dobrej strukturze klasy bardzo naturalnie można wpadać na pomysły jak ją rozwinąć i wykorzystać w dalszym programowaniu.
Oprócz samego pomysłu na strukturę kodu sporo pracy trzeba włożyć w to, żeby inni programiści nie mogli wykorzystać w zły sposób jego funkcji i jednocześnie, żeby metody były na tyle przydatne, żeby zachęciły kogoś do wykorzystania w swoim kodzie i zostawiły miejsce do rozwoju.

Skróty klawiszowe

qwe
asd - mapują odpowiadające graficznie pola gry
zxc

ctrl + s - zaczyna grę

ctrl + 1 – pierwszy gracz ludzki
ctrl + 2 – pierwszy gracz komputer łatwy
ctrl + 3 – pierwszy gracz komputer trudny
ctrl + 4 – drugi gracz ludzki
ctrl + 5 – drugi gracz komputer łatwy
ctrl + 6 – drugi gracz komputer trudny