Міністерство освіти і науки України

Національний технічний університет України

«Київський політехнічний інститут ім. Ігоря Сікорського»

Факультет інформатики та обчислювальної техніки

Кафедра обчислювальної техніки

КУРСОВА РОБОТА

з дисципліни «Інженерія програмного забезпечення»

на тему «Гра Тетріс»

Виконали:

студенти 2 курсу ФІОТ,

група ІО-14

Чаплицький Д. А. З/к №1428

Пономарчук Є. В. З/к №1421

Група ІО-11  
Володін В. В. З/к №1105

Керівник:

Русінов Володимир Володимирович

Київ – 2023

Зміст

[**Вступ** 3](#_Toc136987378)

[**Опис гри, та її складових** 4](#_Toc136987379)

[**Складові гри «блекджек»** 4](#_Toc136987380)

[**Вимоги до гри та функціонал** 5](#_Toc136987381)

[Структура 5](#_Toc136987382)

[Сценарій 6](#_Toc136987383)

[Детальне пояснення 6](#_Toc136987384)

[Імпорт бібліотек: 6](#_Toc136987385)

[Глобальні змінні: 6](#_Toc136987386)

[Клас **Card**: 7](#_Toc136987387)

[Клас **Hand**: 7](#_Toc136987388)

[Функція **generate\_deck()**: 7](#_Toc136987389)

[var\_get(): 8](#_Toc136987390)

[var\_check(): 8](#_Toc136987391)

[check\_values(): 8](#_Toc136987392)

[start\_game(): 8](#_Toc136987393)

[show\_cards\_from\_deal(): 8](#_Toc136987394)

[bid\_button(): 8](#_Toc136987395)

[hit\_button(): 8](#_Toc136987396)

[show\_card(): 9](#_Toc136987397)

[hold(): 9](#_Toc136987398)

[split(): 9](#_Toc136987399)

[double\_button(): 9](#_Toc136987400)

[dealer\_play(): 9](#_Toc136987401)

[dealer\_cards\_reveal(): 10](#_Toc136987402)

[show\_dealer\_cards(): 10](#_Toc136987403)

[Тестування 10](#_Toc136987404)

[generate\_deck(), 10](#_Toc136987405)

[hit(), 10](#_Toc136987406)

[deal(), 10](#_Toc136987407)

[count\_card\_values(). 10](#_Toc136987408)

[Висновок по тестам: 12](#_Toc136987409)

[Код програми 30](#_Toc136987410)

[BJ.py 30](#_Toc136987411)

[Game.py 31](#_Toc136987412)

[Список використаних джерел: 36](#_Toc136987413)

## **Вступ**

Курсова робота виконується на базі знань, отриманих при вивченні курсу

«Основи проектування програмного забезпечення» і загальноосвітніх дисциплін.

**Метою** курсової роботи є закріплення теоретичних знань і практичних навичок

студентів з проектування, моделювання, розробки та тестування програмного

забезпечення.

Змістом курсової роботи є розробка програмного додатку з графічним

інтерфейсом користувача, в моєму випадку – гра «Блекджек»



## **Опис гри, та її складових**

Блекджек - це азартна гра, яка грається проти дилера в казино. Метою гравця є набрати суму очок, яка наближається до 21, але не перевищує її. У грі використовуються звичайна колода з 52 картами.

Кожна карта має свою вартість: числові карти (від 2 до 10) оцінюються за їх числовим значенням, валети, дамі та королі мають вартість 10, а туз може мати значення 1 або 11, залежно від ситуації.

Гравець отримує дві картки, і його мета полягає в тому, щоб набрати більше очок, ніж дилер, але не перебрати 21. Гравець може взяти додаткові картки (хіти) або залишитися при поточній сумі очок (стенд). Після того, як гравці закінчують свої дії, дилер отримує свої картки. Він повинен брати додаткові картки до тих пір, поки його сума очок не стане 17 або більше.

Гра блекджек є популярною через свою простоту та можливість швидкого заробітку, але вимагає від гравця стратегічного мислення та прийняття рішень на основі карт, які були показані.

### **Складові гри «блекджек»**

1. **Карткова колода**: Зазвичай гра в блекджек використовує стандартну колоду з 52 картами без додаткових джокерів. Кожна карта має свою вартість, яка визначається числовим значенням або рангом карти.
2. **Гравець**: У грі можуть брати участь один або більше гравців. Кожен гравець змагається проти дилера (круп'є).
3. **Дилер**: Дилер представляє казино і грає проти гравців. Його завдання - роздавати карти гравцям та виконувати правила гри.
4. **Початок гри**: Гра розпочинається з роздачі двох карт гравцям, які відкриваються лише одна з них. Дилер також отримує дві карти, одна з яких відкривається, а інша залишається закритою.
5. **Ставка**: Перед тим як розпочати гру, кожен гравець робить ставки, які в майбутньому подвоюються(у разі виграшу) або ж спростовуються(у разі програшу) в залежності від дій ігрока: якщо гравець ділить руку(спліт) він повинен підтримати цю дію вхідною ставкою, також можна подвоїти ставку(дабл) – дія каже сама за себе, якщо у гравця перебір або дилер набрав більше очків – гравець втрачає ставку.
6. **Очки**: Кожна карта має свою вартість у очках. Числові карти оцінюються за їх числове значення, картинкові карти (валети, дамі та королі) мають вартість 10, а туз може мати значення 1 або 11, залежно від ситуації.
7. **Рухи гравця**: Гравець має кілька опцій для своїх дій під час свого ходу. Він може взяти ще одну карту (хіт), щоб набрати більше очок, або відмовитися від додаткових карт і залишитися з поточною сумою очок (холд). Також можуть бути доступні інші опції, такі як подвоєння ставки або розділення пари карт на дві руки.
8. **Перебор**: Якщо сума очок гравця перевищує 21, він перебирає і втрачає ставку.
9. **Виграш і програш**: Після того, як всі гравці закінчили свої дії, дилер розкриває свою закриту карту. Він повинен брати додаткові картки до суми очок 17 або більше. Потім порівнюються суми очок гравців і дилера. Якщо сума очок гравця більша за суму очок дилера і не перевищує 21, гравець перемагає і отримує виграш у вигляді своєї ставки. Якщо сума очок дилера більша або рівна сумі очок гравця, гравець програє і втрачає свою ставку.
10. **Кінцева умова**: Гравець вибуває з гри, якщо на його балансі не залишається грошей(очків, фішок, тощо.) для ставки.

## **Вимоги до гри та функціонал**

1. **Роздача карт**: Система повинна здійснювати роздачу карт гравцю та дилеру відповідно до правил гри блекджек.
2. **Обчислення очків**: Система повинна розраховувати суму очок для гравця і дилера на основі отриманих карт.
3. **Взяття додаткових карт**: Гравець повинен мати можливість брати додаткові карти (хіти) з метою набрати ближче до 21 очка.
4. **Прийняття рішень гравцем**: Гравець повинен мати можливість приймати рішення щодо своїх дій, таких як хіт, дабл, спліт або холд.
5. **Розділення пари карт** (опціонально): Якщо гравець отримує пару однакових карт, система може надати опцію розділення пари на дві руки та гру з ними окремо.
6. **Підрахунок виграшу**: Система повинна визначати результат гри, порівнюючи суму очок гравця і дилера та враховуючи правила виграшу та програшу.
7. **Подвоєння ставки**: Гравець повинен мати можливість подвоїти свою ставку тільки після роздачі карт
8. **Відображення гри**: Гра може мати графічний інтерфейс або текстове відображення, щоб гравець міг бачити свої карти, суму очок та рухи дилера.
9. **Можливість нової гри**: Після закінчення поточної гри гравець повинен мати можливість розпочати нову гру з новою роздачею карт.

### Структура

Ми розробляли гру блекджек на мові програмування Python за допомогою бібліотеки ‘Tkinter’.

Основні класи, які використовувались в коді гри:

**Hand** – безпосередньо рука дилера або гравця(в нашій адаптації може бути до 4 включно).

**Card** – клас, що описує гральні карти.

**Game** – клас, що описує інтерфейс гри

**BJ** – клас, у якому описується ігрова логіка

### Сценарій

1. Роздача карт: Гравець робить ставку, після чого дилер роздає по дві картки і гравцеві, і собі. Одна з карт дилера залишається закритою.
2. Рухи гравця: Гравець робить свої рухи, вибираючи одну з наступних опцій:
   1. Хіт (Hit): Гравець бере ще одну карту.
   2. Дабл(Double): Одразу після роздачі гравець може подвоїти свою ставку і взяти ще одну карту.  
       Спліт(Split): Якщо у гравця в руці тільки 2 карти і вони однакові, він має можливість розділити цю руку на 2, добираючи по 1 карті в кожну з них.
   3. Стенд (Stand): Гравець відмовляється від додаткових карт і залишається з поточною сумою очок.
3. Хід дилера: Після того, як гравець закінчив свої рухи, дилер відкриває свою закриту карту і робить свої ходи згідно з правилами гри блекджек. Дилер бере додаткові карти до тих пір, поки його сума очок не стане 17 або більше.
4. Порівняння результатів: Після того, як дилер закінчив свої ходи, порівнюються суми очок гравця і дилера. Результати можуть бути наступними:
   1. Якщо сума очок гравця більша за суму очок дилера і не перевищує 21, гравець перемагає і отримує виграш, зазвичай в розмірі своєї ставки.
   2. Якщо сума очок гравця менша або рівна сумі очок дилера, гравець програє і втрачає свою ставку.
   3. Якщо суми очок гравця і дилера однакові (нічия), ставка може повертатися гравцю або вважатися "push" без виграшу або програшу.
   4. Нова гра: Після вирішення результатів гравець має можливість розпочати нову гру з новою роздачею карт, або може залишити гру.

# Детальне пояснення

### Імпорт бібліотек:

* + **from tkinter import PhotoImage** імпортує клас **PhotoImage** з бібліотеки tkinter.
  + **import random** імпортує модуль **random**, який буде використовуватися для отримання випадкових значень.

### Глобальні змінні:

* + **hands\_counter = 0** - лічильник рук, який використовується для відстеження кількості рук гравця.
  + **hands = []** - список рук гравця.
  + **deck = []** - колода карт.

### Клас **Card**:

* + Клас представляє окрему карту у грі.
  + **\_\_init\_\_(self, card\_suit, value, card\_sign)** - конструктор класу, приймає параметри:
    - **card\_suit** - масть карти (наприклад, "spades", "clubs", "hearts", "diamonds").
    - **value** - числова вартість карти.
    - **card\_sign** - символ або значення карти (наприклад, "A", "2", "3", ..., "K").
  + У конструкторі також створюється об'єкт **PhotoImage**, який відповідає графічному зображенню карти.

### Клас **Hand**:

* + Клас представляє руку гравця у грі.
  + **\_\_init\_\_(self, first\_value, second\_value)** - конструктор класу, приймає параметри:
    - **first\_value** - перша вартість карт у руці.
    - **second\_value** - друга вартість карт у руці.
  + Клас містить деякі атрибути, такі як **cards\_in\_hand** (список карт у руці), **actual\_bid** (поточна ставка) та **status** (стан руки).
  + **count\_card\_values(self)** - метод для підрахунку вартості карт у руці гравця.
  + **hit(self)** - метод, який додає нову карту з колоди до руки гравця.
  + **deal(self)** - метод, який роздає гравцю початкові дві карти.

### Функція **generate\_deck()**:

* + Генерує і повертає нову колоду карт.
  + Використовується список **card\_suits**, що містить масті ("spades", "clubs", "hearts", "diamonds").
  + Використовується список **card\_values**, що містить числові значення карт.
  + Використовується список **card\_signs**, що містить символи або значення карт.
  + Виконується подвійний цикл для створення об'єктів **Card** з усіх можливих комбінацій мастей та значень.
  + Згенеровані карти додаються до колоди **deck**.

var\_get(): Ця функція отримує значення змінної var і зберігає його в глобальній змінній hand\_number.

var\_check(): Ця функція перевіряє статус різних "рук" (гравців) і встановлює значення var відповідно до статусу активної "руки". Якщо всі руки, окрім дилера, неактивні, викликається функція dealer\_play(). В іншому випадку встановлюється значення var для активної "руки".

check\_values(): Ця функція перевіряє значення карти для поточної "руки" (hand\_number). Якщо сума значень карт дорівнює 21, статус "руки" встановлюється як "неактивна", а відповідний віджет оновлюється. Якщо сума значень перевищує 21, відповідний віджет оновлюється, викликається функція hold(), а потім перевіряється кількість рук. Якщо кількість рук дорівнює 2, викликається функція dealer\_play(). В іншому випадку викликається функція var\_check().

start\_game(): Ця функція розпочинає гру. Вона ініціалізує глобальну змінну deck, додає дві "руки" (Hand) до списку hands і встановлює початкову ставку для другої "руки". Потім генерується колода карт, карти роздаються на "руки", і викликається функція show\_cards\_from\_deal().

show\_cards\_from\_deal(): Ця функція відображає картки з роздачі на ігровому інтерфейсі. Вона створює різні віджети (Canvas, Label, Radiobutton) та розміщує їх на відповідних позиціях в графічному інтерфейсі. Картки гравців відображаються на віджетах з використанням зображень, а також встановлюються деякі значення змінних (var, hand\_number). Після цього викликається функція check\_values() для перевірки значень карт поточної "руки".

bid\_button(): Ця функція обробляє натискання кнопки "BID" на ігровому інтерфейсі. Вона отримує значення ставки (bid) з текстового поля введення і перевіряє його на відповідність обмеженням (більше балансу гравця або менше 0). Якщо ставка є прийнятною, кнопки для дій у грі активуються, баланс оновлюється, текстове поле та кнопка "BID" видаляються, і запускається функція start\_game(). У разі неприпустимої ставки видається спливаюче повідомлення.

hit\_button(): Ця функція обробляє натискання кнопки "HIT" на ігровому інтерфейсі. Вона викликає функцію var\_get() для отримання значення hand\_number, додає нову карту до поточної "руки" гравця (hands[hand\_number].hit()), оновлює відображення карт на інтерфейсі (show\_card()), і перевіряє значення карт поточної "руки" (check\_values()).

show\_card(): Ця функція відображає нову карту на ігровому інтерфейсі. Вона створює новий віджет Canvas та розміщує його на відповідній позиції в графічному інтерфейсі. Зображення нової карти встановлюється на цей віджет.

hold(): Ця функція обробляє натискання кнопки "HOLD" на ігровому інтерфейсі. Вона викликає функцію var\_get() для отримання значення hand\_number, змінює статус поточної "руки" на "неактивна" (hands[hand\_number].status = "Inactive"), вимикає віджети, пов'язані з поточною "рукою" (widgets[hand\_number \* 3]["state"] = DISABLED), оновлює значення другого значення карт у віджеті (widgets[hand\_number \* 3 + 1]["text"] = hands[hand\_number].second\_value), і викликає функцію var\_check().

split(): Ця функція обробляє натискання кнопки "SPLIT" на ігровому інтерфейсі. Вона викликає функцію var\_get() для отримання значення hand\_number, перевіряє можливість розділення поточної "руки" (два однакові значення карт, досить коштів на балансі гравця), розділяє поточну "руку" на дві нові "руки" (hands.append(Hand(0, 0))), оновлює баланс гравця, видаляє віджети, пов'язані з поточною "рукою", створює нові віджети для нових "рук", додає карту до кожної нової "руки", перераховує значення карт у нових "руках", викликає функцію check\_values(), оновлює значення змінної var, і викликає функцію var\_get() та check\_values().

double\_button(): Ця функція обробляє натискання кнопки "DOUBLE" на ігровому інтерфейсі. Вона викликає функцію var\_get() для отримання значення hand\_number, перевіряє можливість подвоєння ставки (дві карти у поточній "руці", достатньо коштів на балансі гравця), збільшує ставку на поточній "руці", оновлює баланс гравця, викликає функцію hit() для поточної "руки", вимикає віджети, пов'язані з поточною "рукою", оновлює відображення карт поточної "руки", перевіряє значення карт (check\_values()), і викликає функцію var\_check().

dealer\_play(): Ця функція відображає дії дилера під час гри. Вона показує карти дилера, визначає значення карт дилера згідно правил гри у блекджек, порівнює значення карт гравця та дилера, оновлює баланс гравця, виводить сповіщення з поточним балансом гравця, очікує 1 секунду, перевіряє, чи баланс гравця дорівнює 0, і якщо так, то викликає root.quit(), оновлює список widgets, змінну hand\_number, ігрові "руки" (hands), видаляє існуючі віджети, створює нові віджети для дилера і гравця, вимикає віджети button\_split, button\_double, button\_hold і button\_hit, створює кнопку button\_bid та поле введення entry\_of\_bid.

dealer\_cards\_reveal(): Ця функція відображає карти дилера, коли він розкриває свою першу карту. Вона знищує віджет дилера, створює новий віджет дилера, відображає першу і другу карту дилера.

show\_dealer\_cards(): Ця функція відображає карту дилера після кожного ходу. Вона відображає останню карту дилера на віджеті дилера.

# Тестування

Тестування коду було виконано за допомогою юніт-тестів. Ми протестували такі функції та методи, як:

## generate\_deck(),

Перевіряли кількість карт які згенерувалися

## hit(),

Перевіряли чи видаляється карта з колоди і чи додається вона до руки

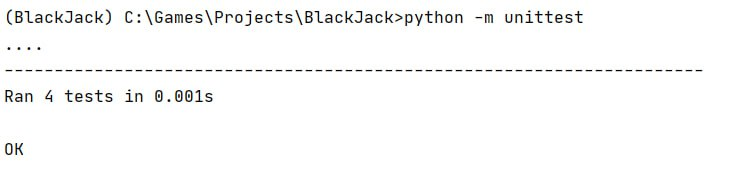
## deal(),

Теж саме що і хіт.

## count\_card\_values().

Перевіряли як рахуються значення карт у руці.

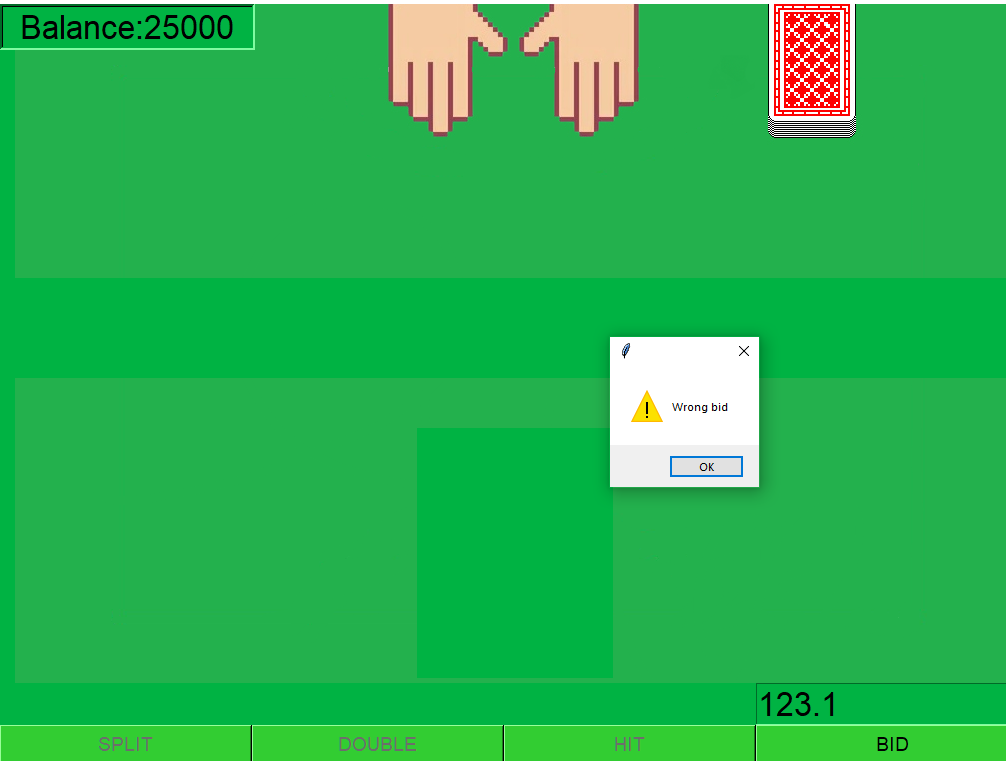
import unittest as u  
from bj import \*  
  
  
class HandClassTest(u.TestCase):  
 def test\_deal(self):  
 hand1 = Hand(0, 0)  
 deck = generate\_deck()  
 hand1.deal()  
 self.assertEqual(len(hand1.cards\_in\_hand), 2)  
 self.assertEqual(len(deck), 50)  
  
 def test\_generate\_deck(self):  
 deck = generate\_deck()  
 self.assertEqual(len(deck), 52)  
  
 def test\_hit(self):  
 hand1 = Hand(0, 0)  
 deck = generate\_deck()  
 hand1.hit()  
 self.assertEqual(len(hand1.cards\_in\_hand), 1)  
 self.assertEqual(len(deck), 51)  
  
 def test\_count\_card\_values(self):  
 hand1 = Hand(0, 0)  
 deck = generate\_deck()  
 for i in deck:  
 if i.card\_sign.\_\_eq\_\_(**"A"**):  
 hand1.cards\_in\_hand.append(i)  
 deck.remove(i)  
 break  
 hand1.count\_card\_values()  
 self.assertEqual(hand1.first\_value, 1)  
 self.assertEqual(hand1.second\_value, 11)  
 for i in deck:  
 if i.card\_sign.\_\_eq\_\_(**"8"**):  
 hand1.cards\_in\_hand.append(i)  
 deck.remove(i)  
 break  
 hand1.count\_card\_values()  
 self.assertEqual(hand1.first\_value, 9)  
 self.assertEqual(hand1.second\_value, 19)  
 hand2 = Hand(0, 0)  
 deck = generate\_deck()  
 for i in deck:  
 if i.card\_sign.\_\_eq\_\_(**"8"**):  
 hand2.cards\_in\_hand.append(i)  
 deck.remove(i)  
 break  
 hand2.count\_card\_values()  
 self.assertEqual(hand2.first\_value, 8)  
 self.assertEqual(hand2.second\_value, 8)  
 for i in deck:  
 if i.card\_sign.\_\_eq\_\_(**"A"**):  
 hand2.cards\_in\_hand.append(i)  
 deck.remove(i)  
 break  
 hand2.count\_card\_values()  
 self.assertEqual(hand2.first\_value, 9)  
 self.assertEqual(hand2.second\_value, 19)  
 hand3 = Hand(0, 0)  
 deck = generate\_deck()  
 for i in deck:  
 if i.card\_sign.\_\_eq\_\_(**"K"**):  
 hand3.cards\_in\_hand.append(i)  
 deck.remove(i)  
 break  
 hand3.count\_card\_values()  
 self.assertEqual(hand3.first\_value, 10)  
 self.assertEqual(hand3.second\_value, 10)  
 for i in deck:  
 if i.card\_sign.\_\_eq\_\_(**"A"**):  
 hand3.cards\_in\_hand.append(i)  
 deck.remove(i)  
 break  
 hand3.count\_card\_values()  
 self.assertEqual(hand3.first\_value, 21)  
 self.assertEqual(hand3.second\_value, 21)

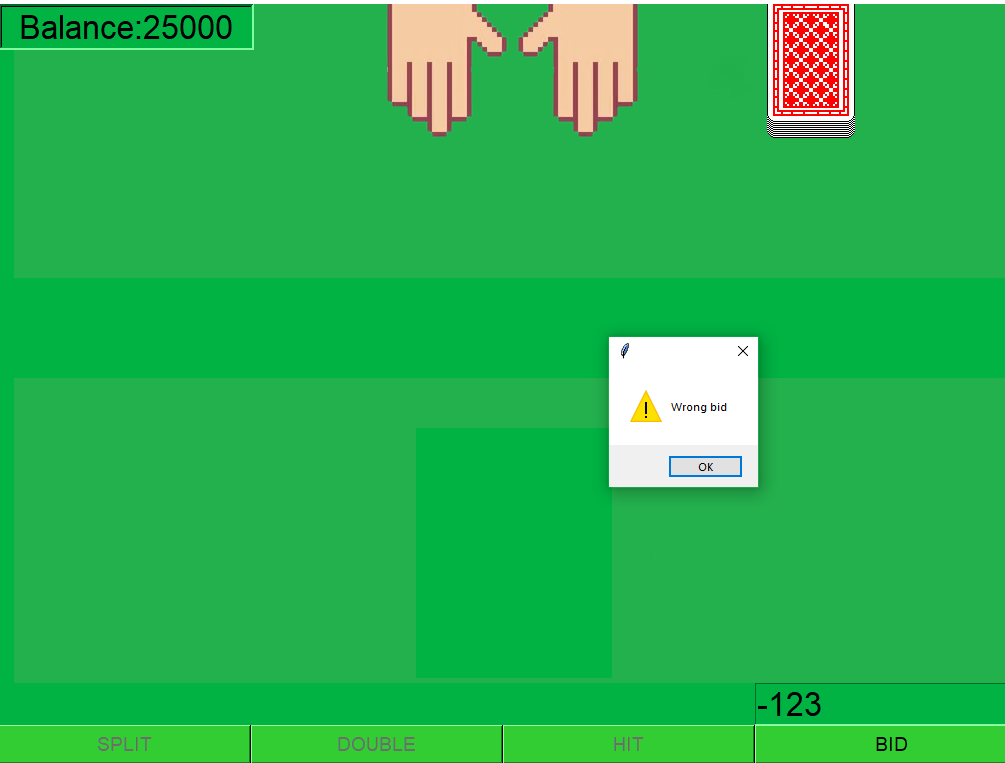


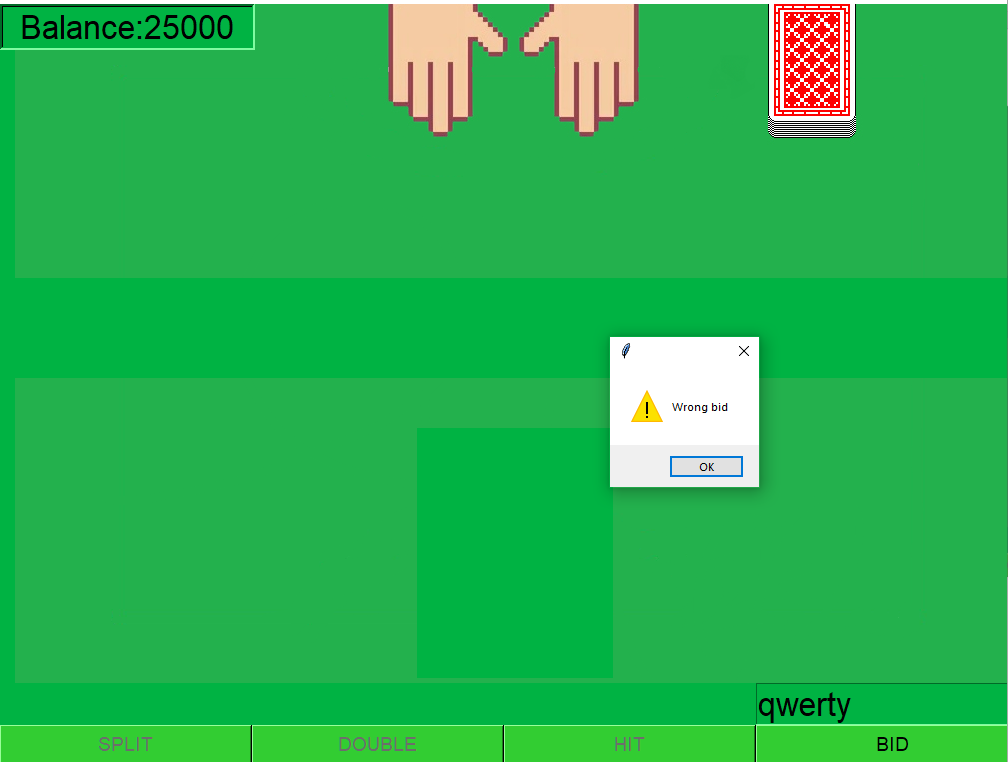
## Висновок по тестам:

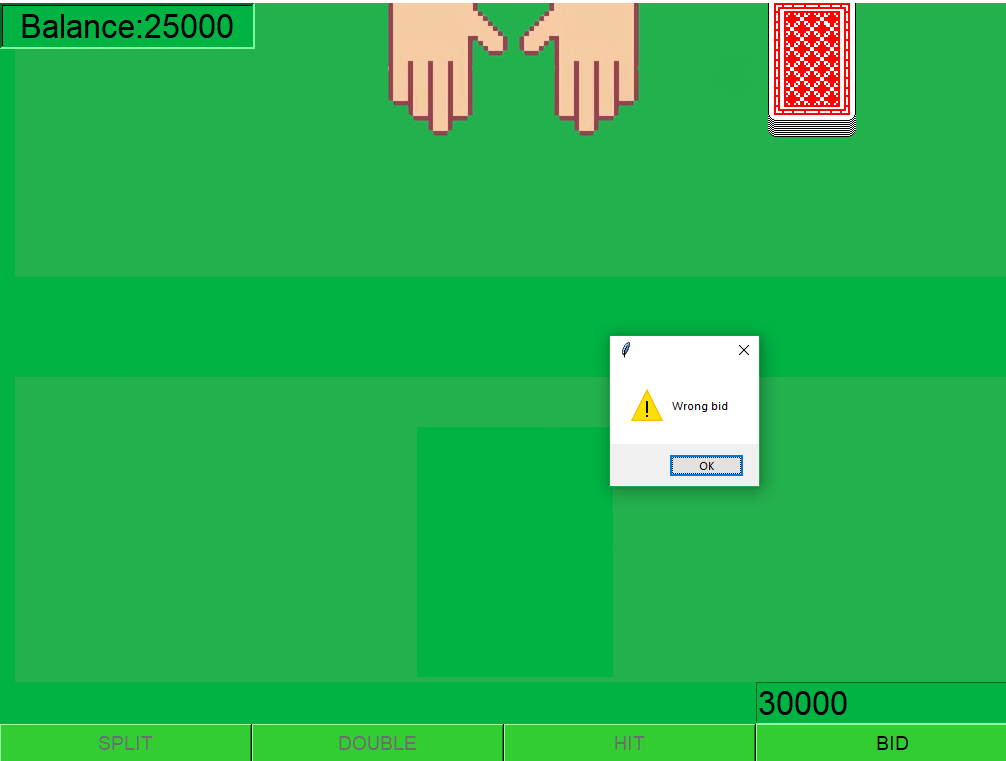
Проведене тестування показало, що всі юніт-тести пройшли успішно і не виявили помилок  
  
Скріншоти та геймплей(тестування):

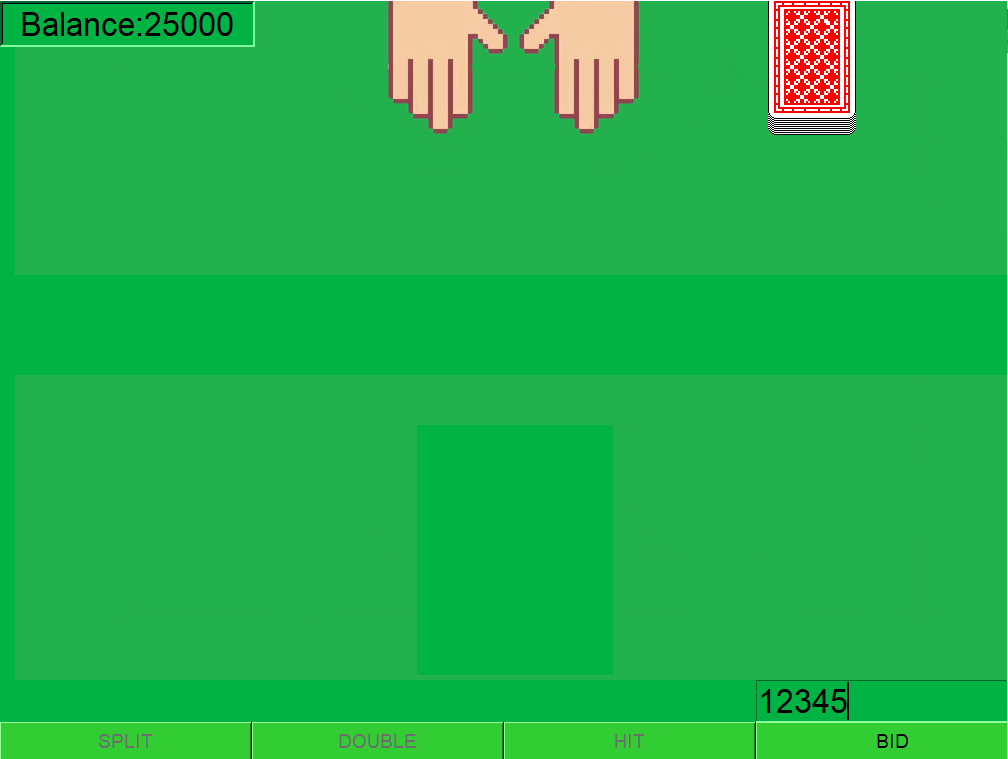
Перевіримо вводу різних чисел для ставки:

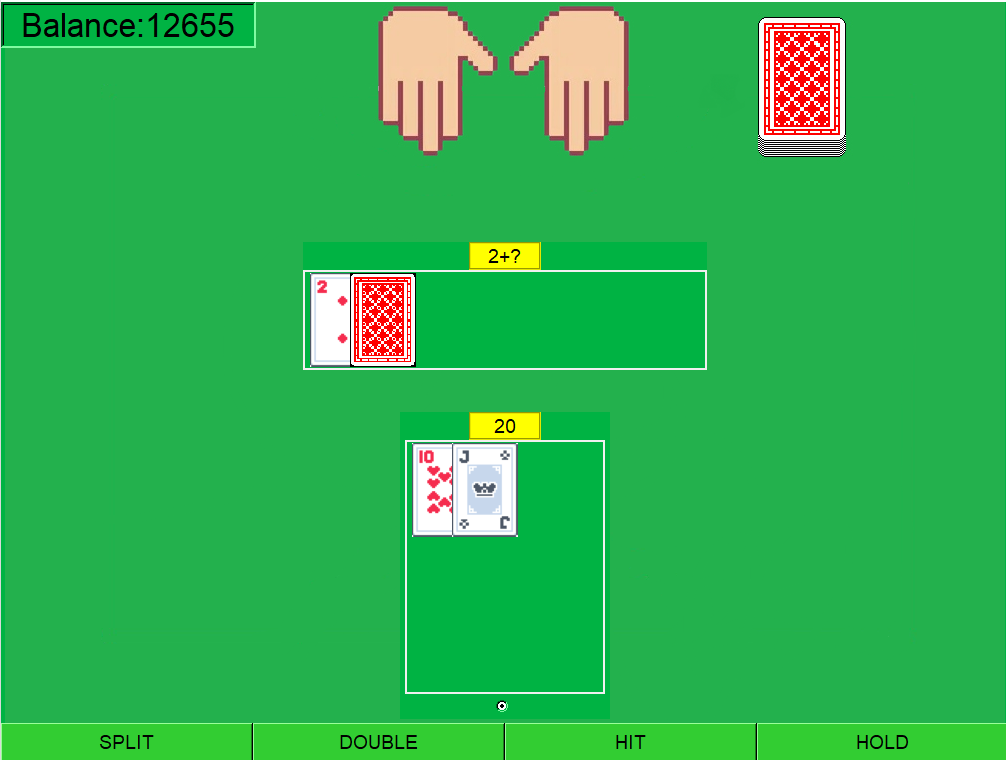






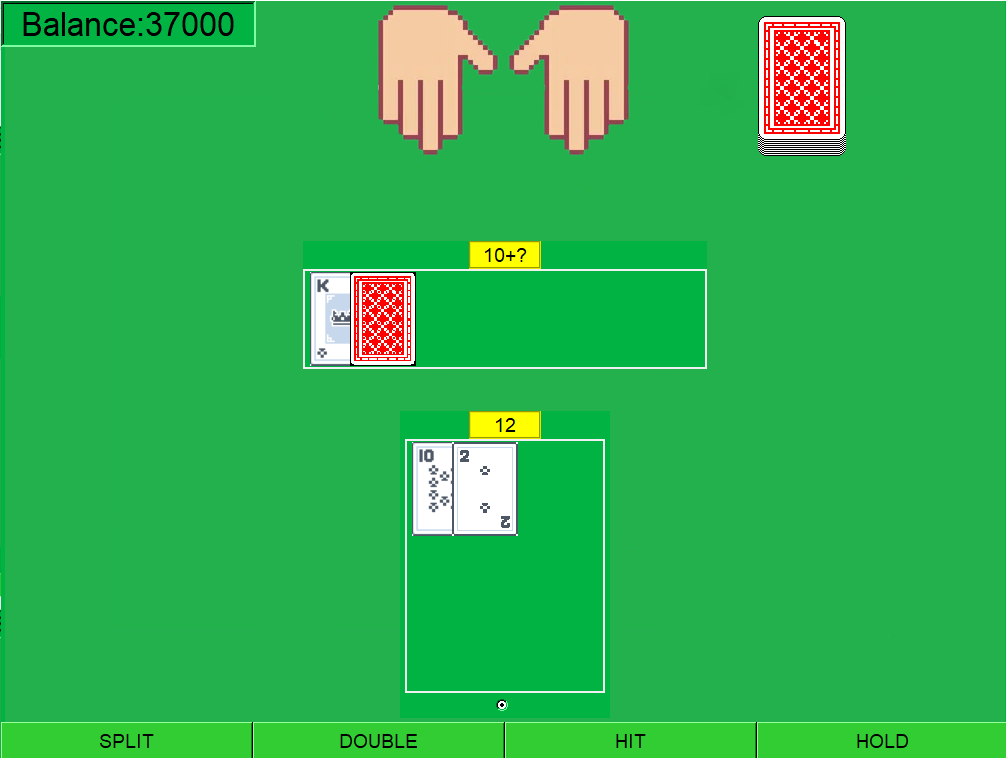


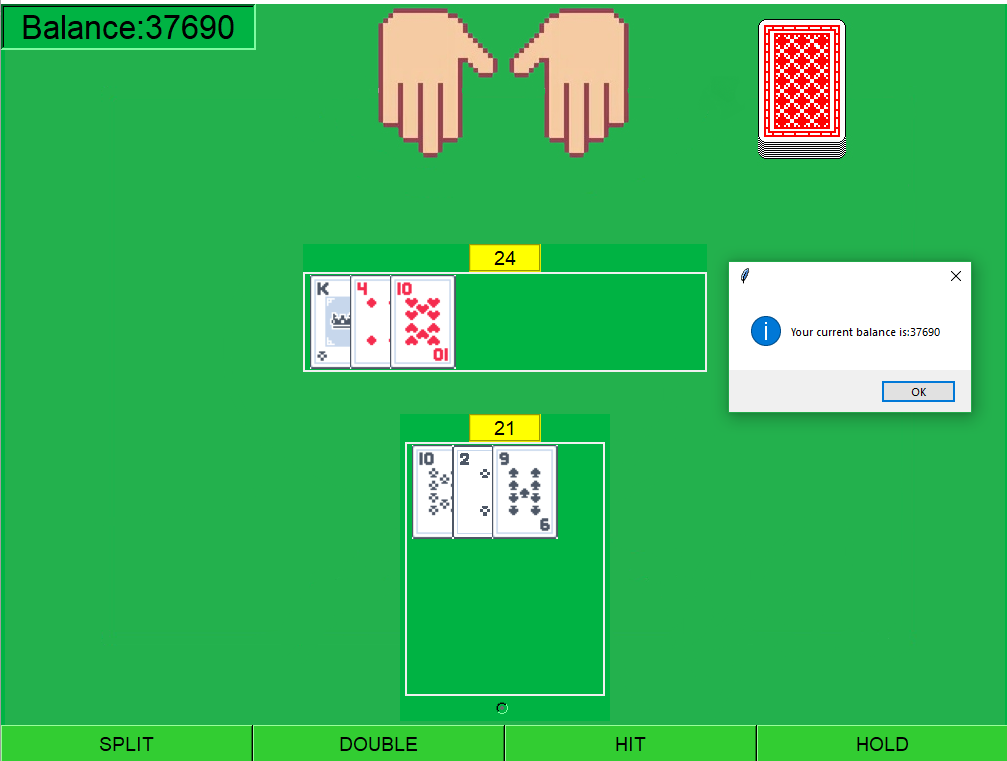




Після того, як ми ввели значення допустимого типу і не більше ніж поточний баланс гравця.

Перевіримо кнопку “HIT”, після нитиску на яку нам має видатися ще одна карту в поточну руку(,яка визначається Radiobutton-ом під кожним полем з картами)

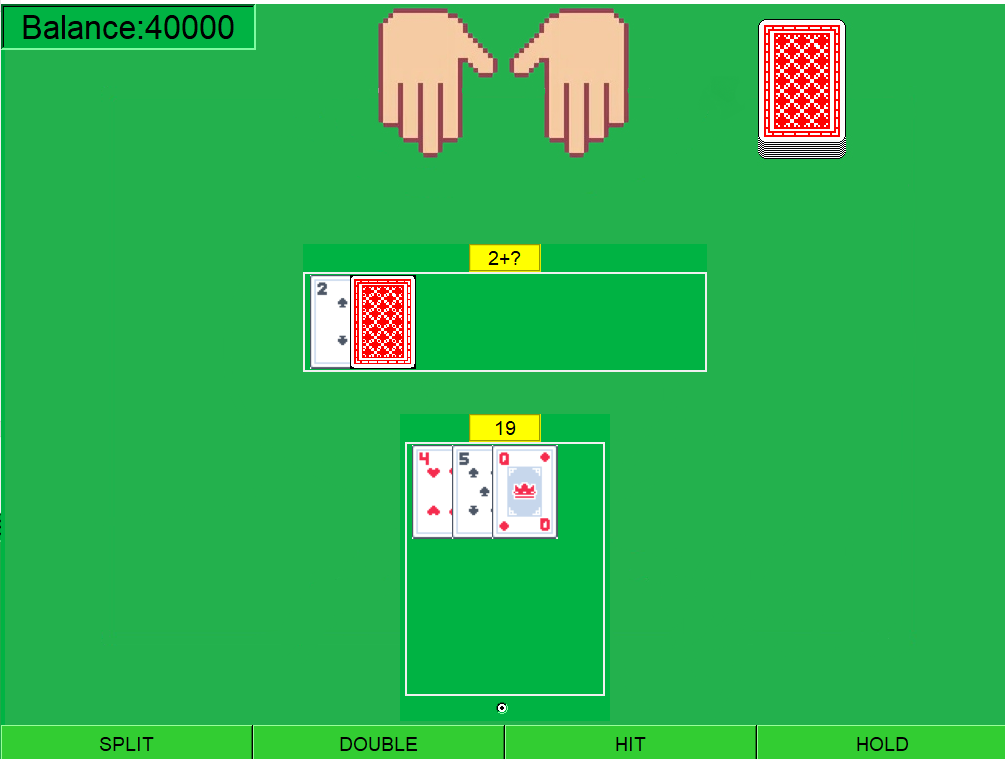




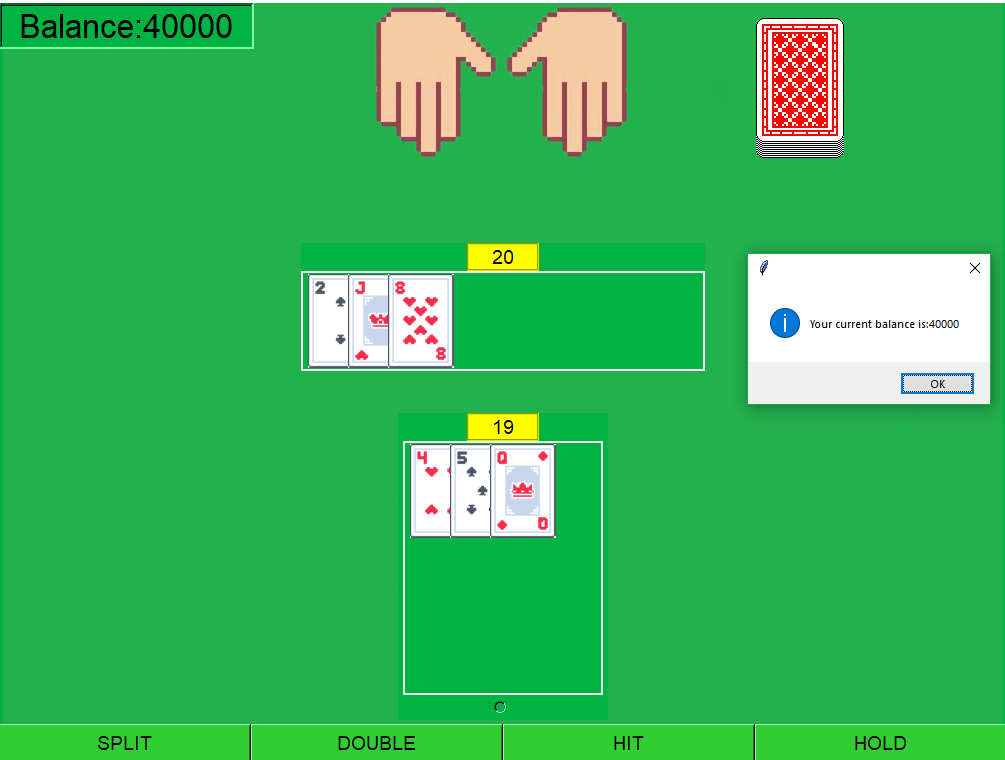
В данному випадку ми отримали значення 21, що означає автоматично, що нам не треба більше нічого в руку, після цього починає грати ділер доти, доки не матиме 17 очок чи більше на руках.

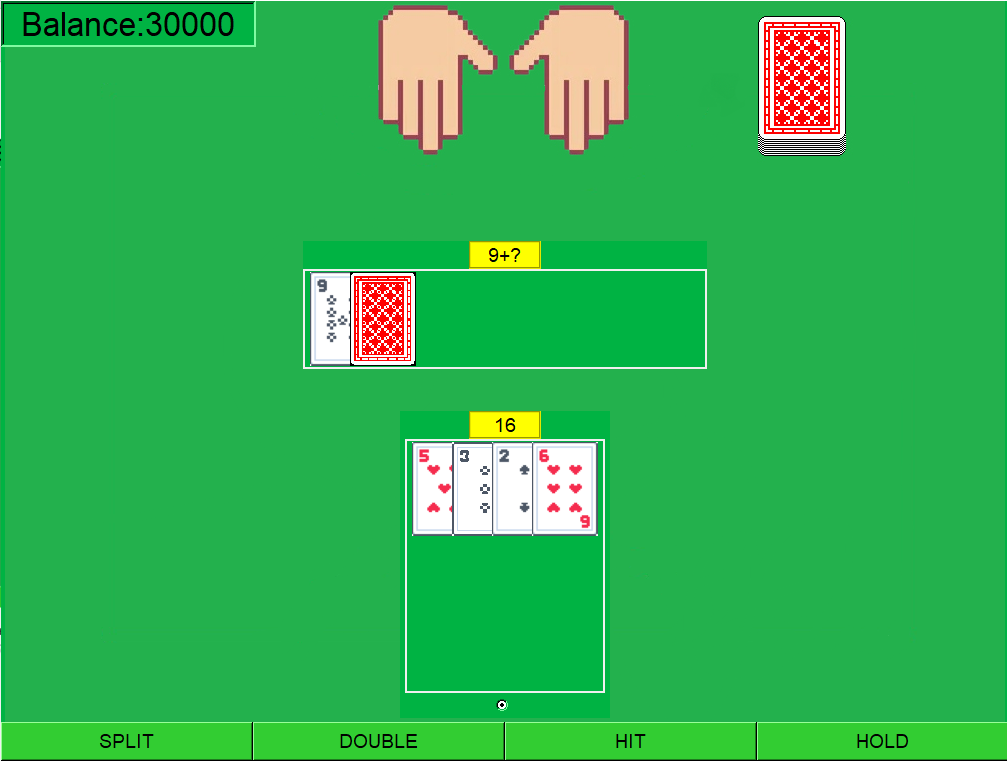
Після того як ділер завершив свою гру, нам виводиться повідомлення з поточним балансом після виграшу чи програшу. Після цього виводиться поле для задання нової ставки.

Неведемо більше прикладів з натисканням на кнопку “HIT”

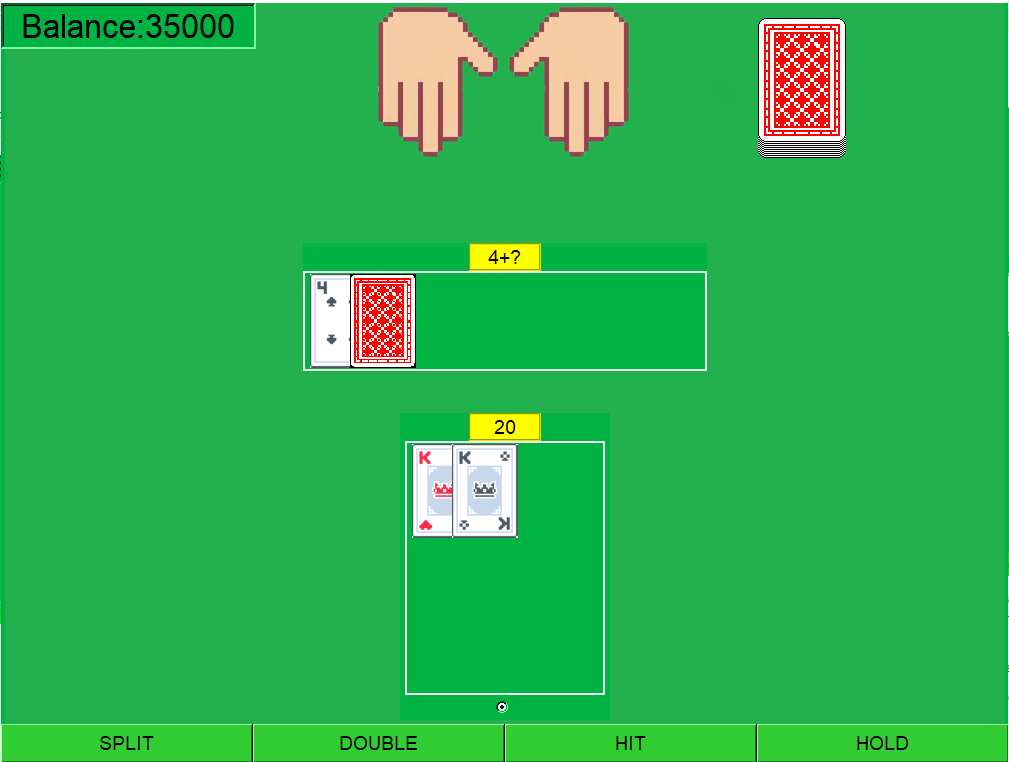


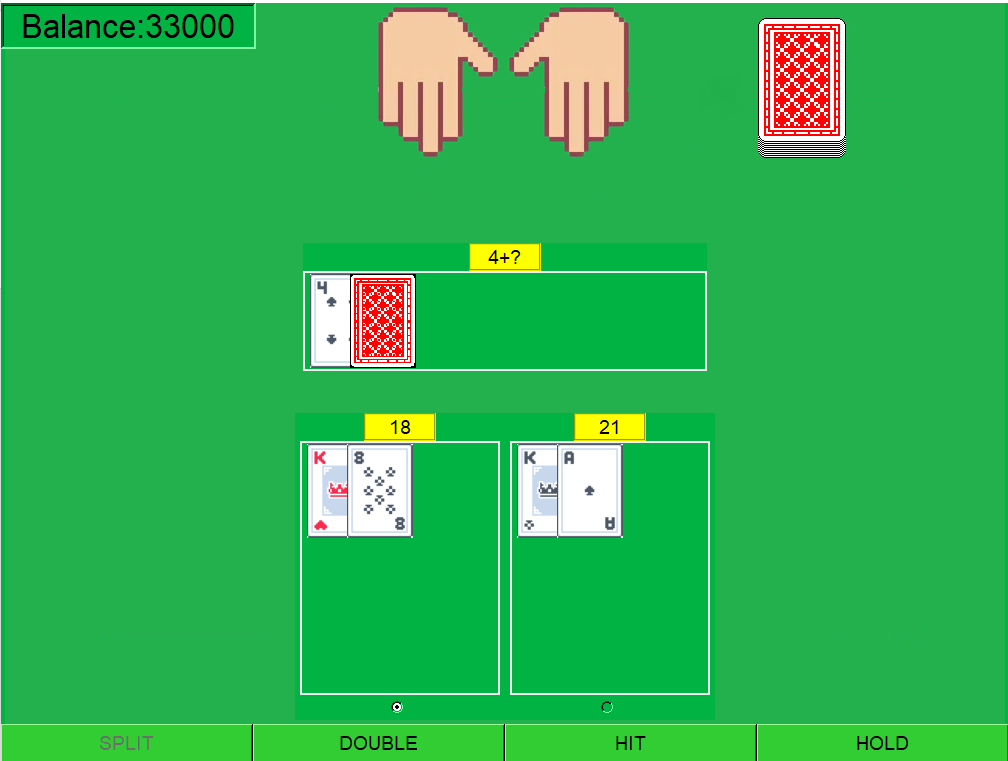
В цьому випадку ми можемо натиснути ще раз або натиснути кнопку “HOLD”, після натискання якої поточна рука стане неактивною.

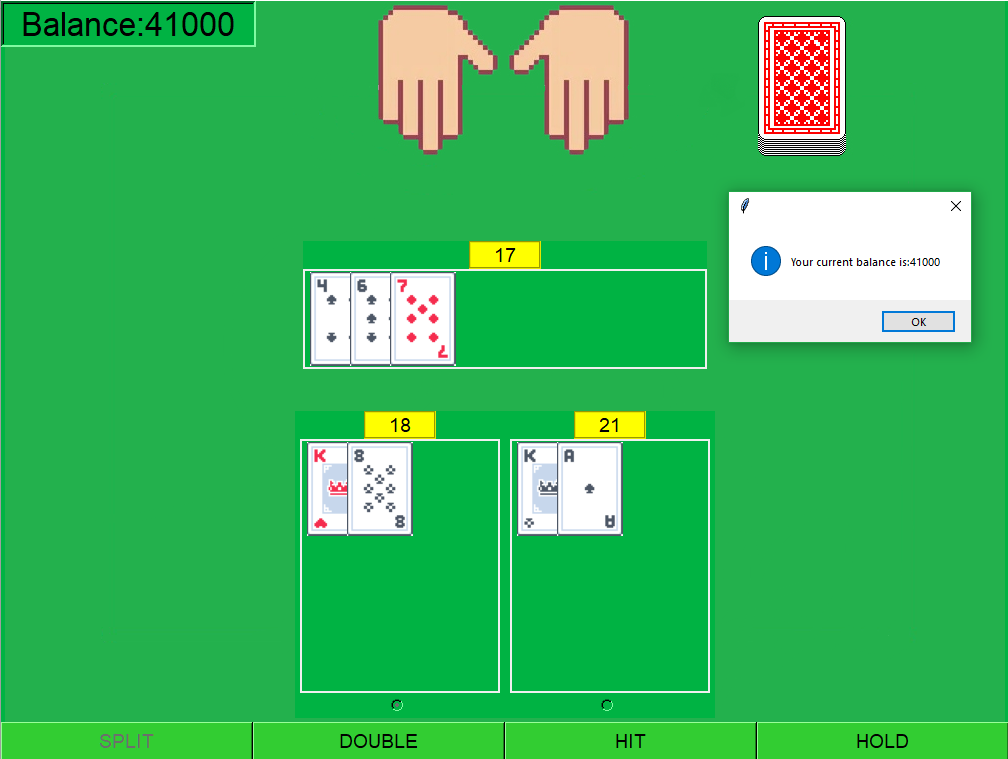


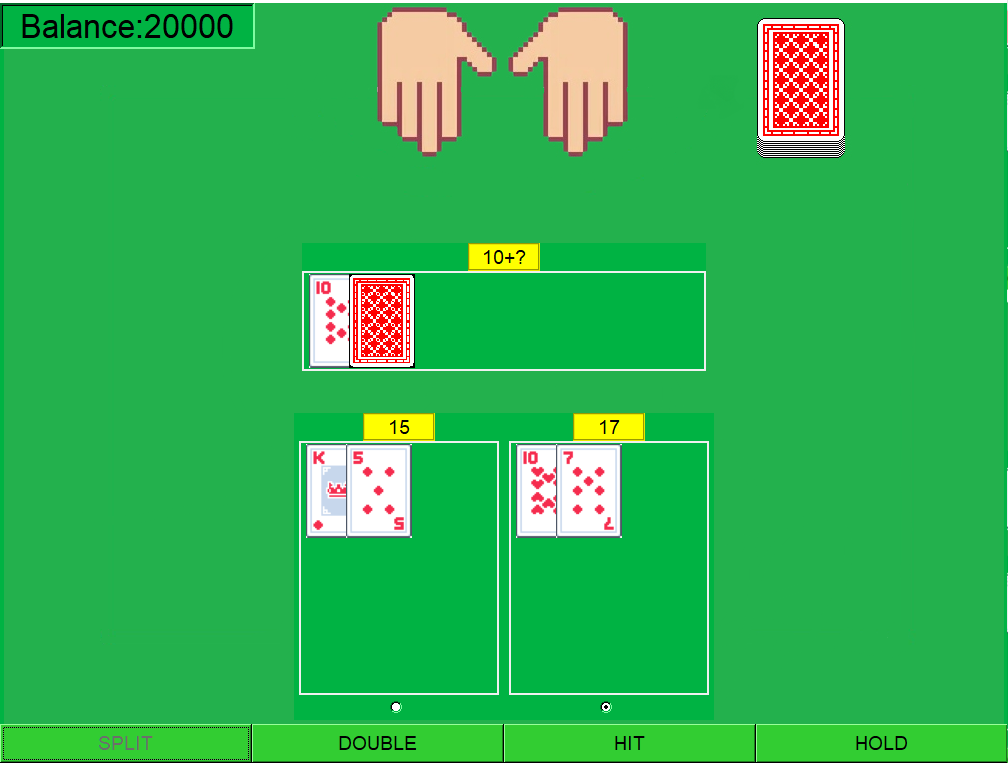


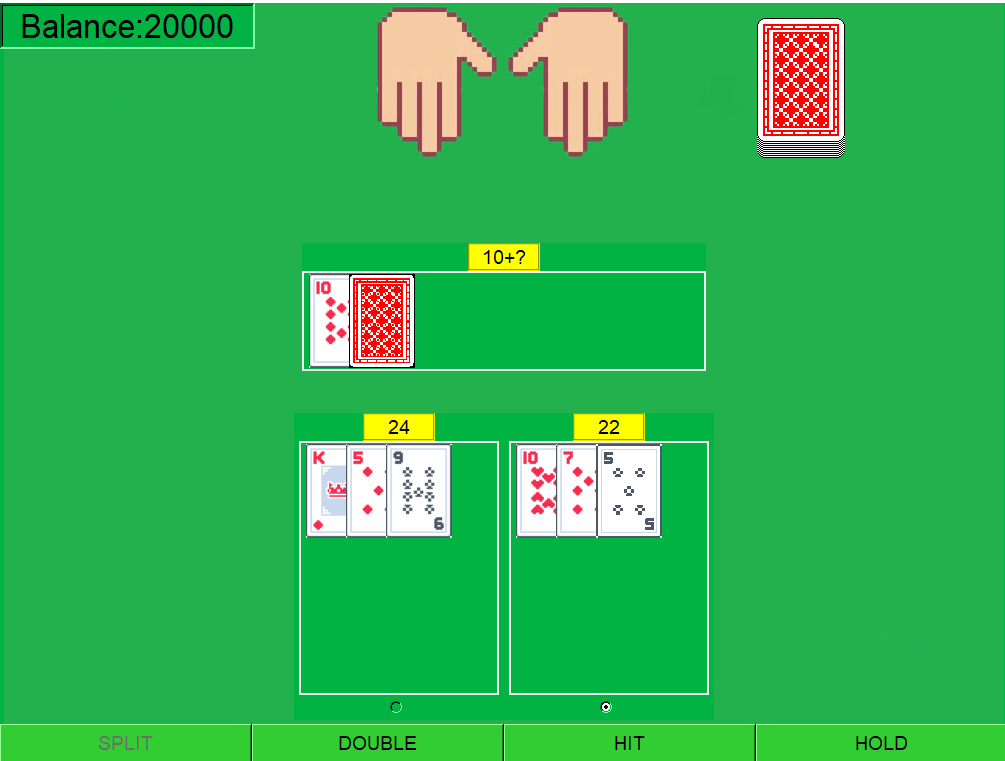
Перевіримо роботу кнопки “SPLIT”. Вона буде працювати а тому випадку, якщо в першій руці дві карти одного номіналу. Функція кнопки “SPLIT” в розділі однієї руки на дві, притому ставка у кожної руки тепер однакові, які ми задавали при початку роздачі. Після натиску кнопки “SPLIT” в кожну руку виділяється по одній карті з першої і додається по одній з колоди, після цього руки стають незалежними



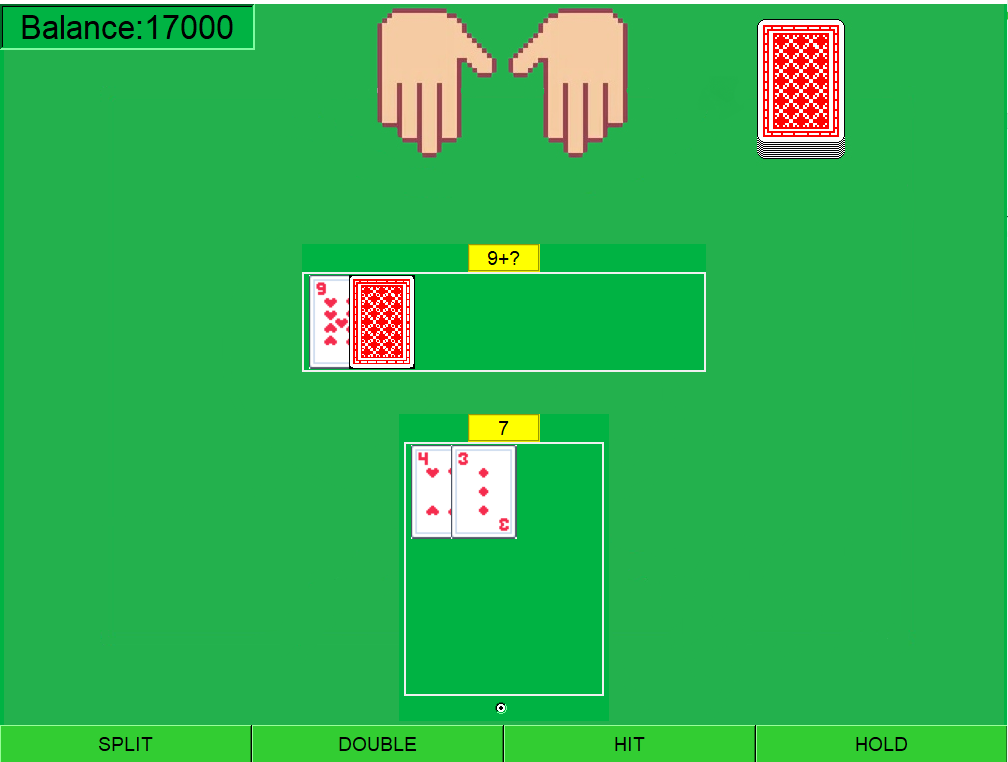


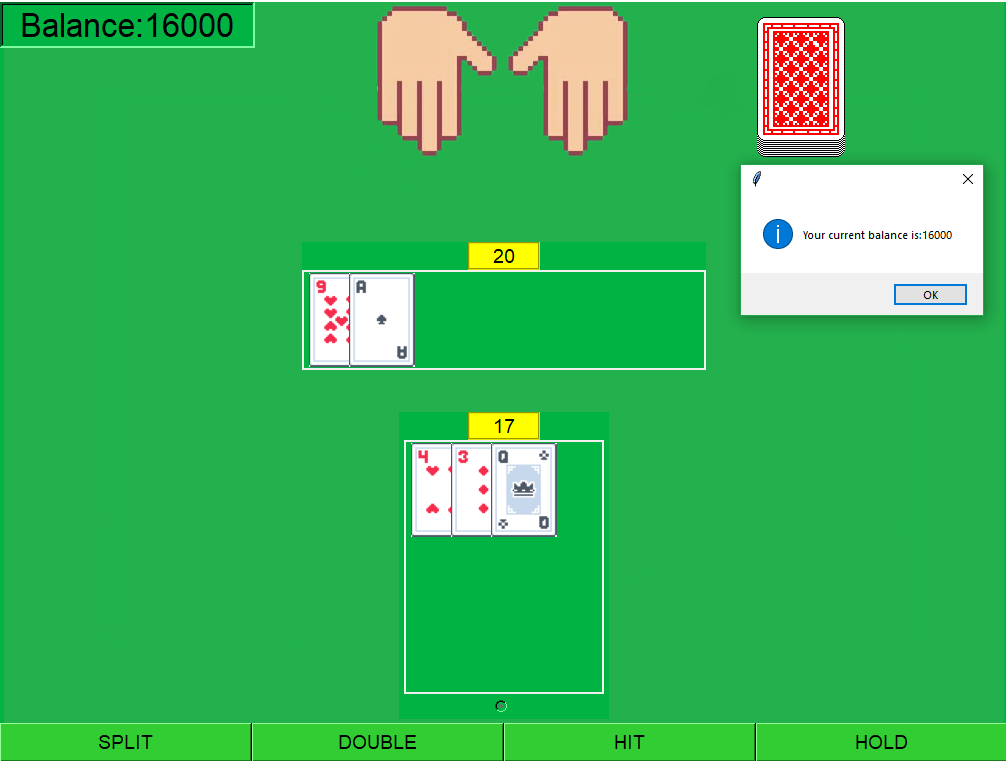




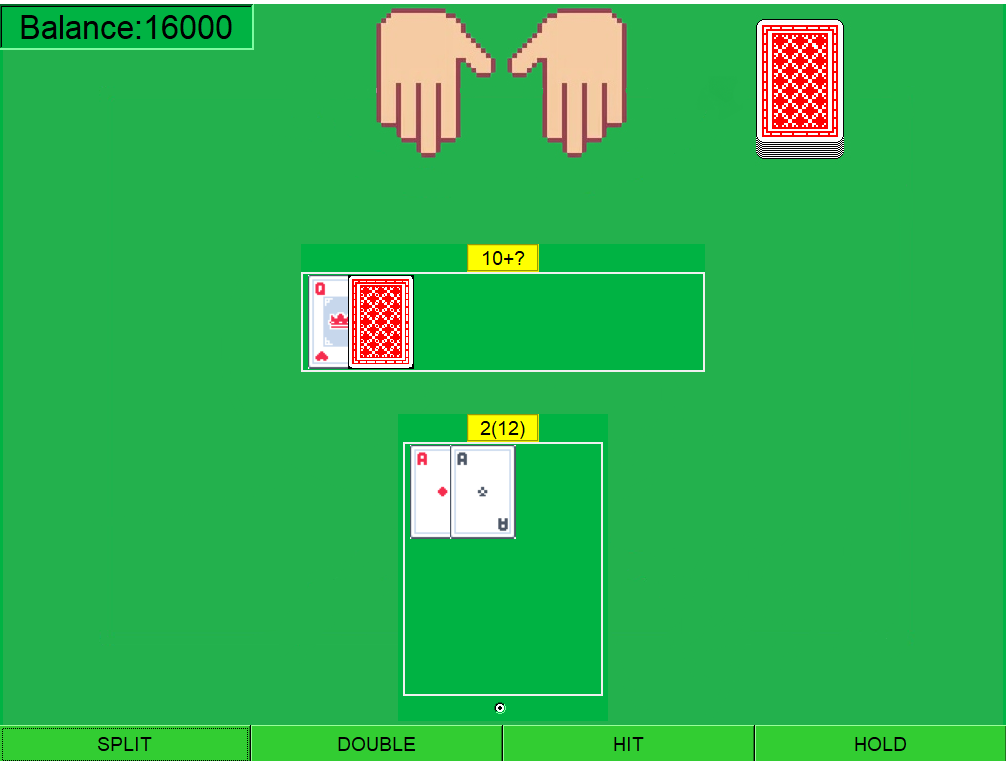


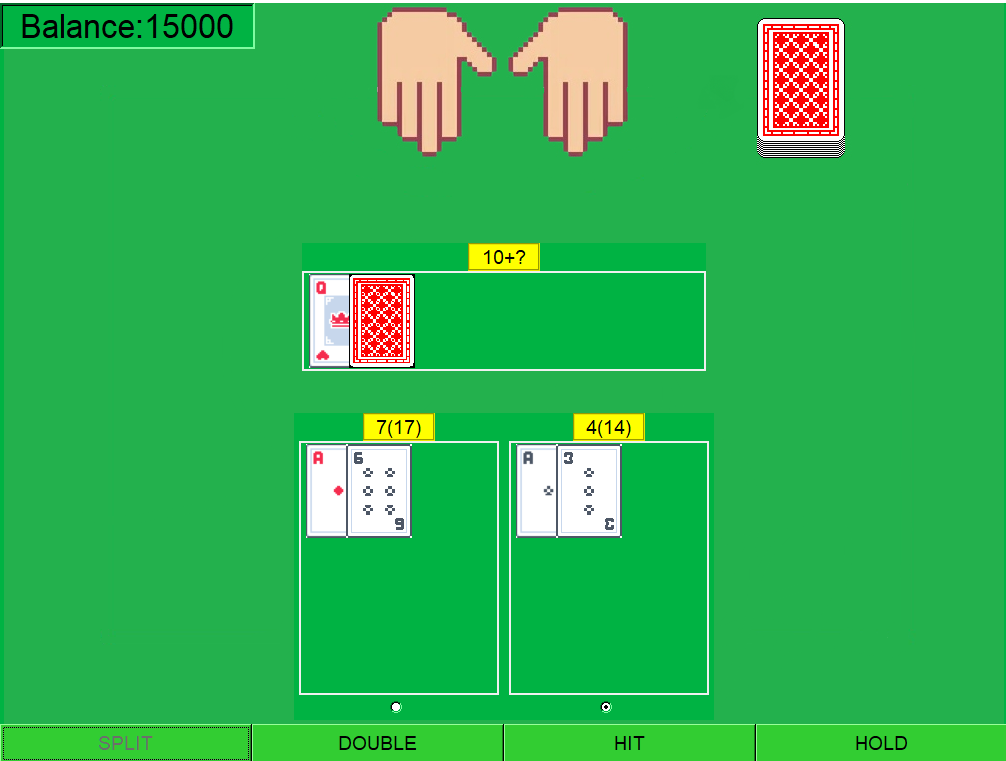
Перевіримо роботу кнопки “DOUBLE”. Вона буде працювати, якщо у вас в поточній руці 2 карти і не більше. Після цього поточна ставка для данної руки подвоюється і рука стає неактивна.

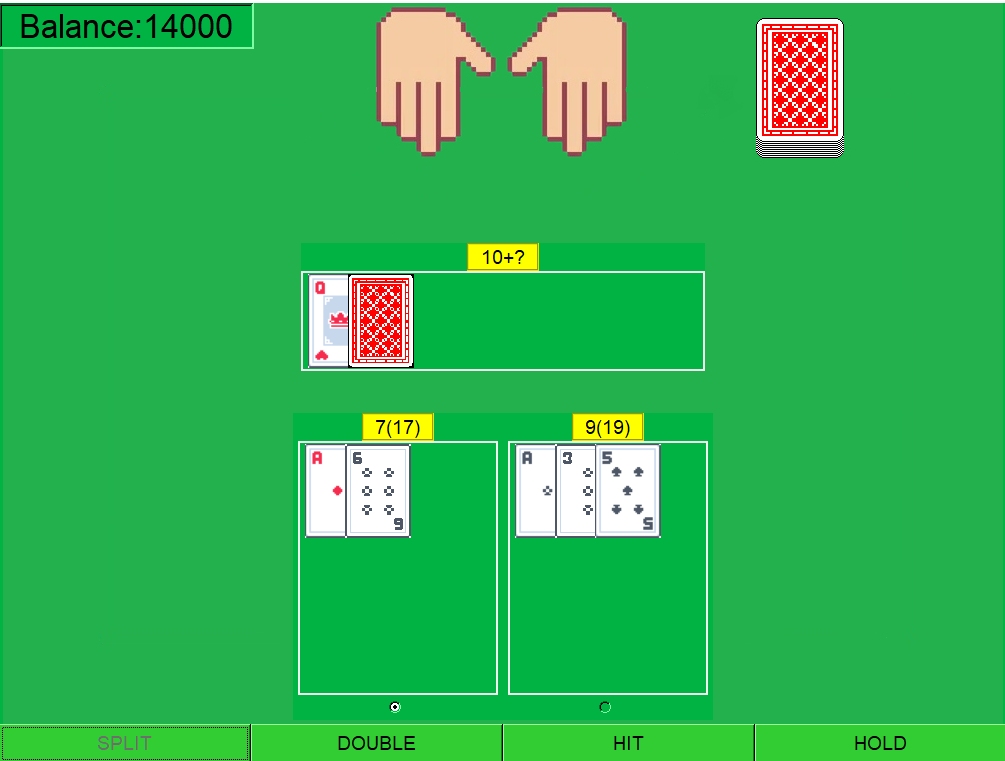


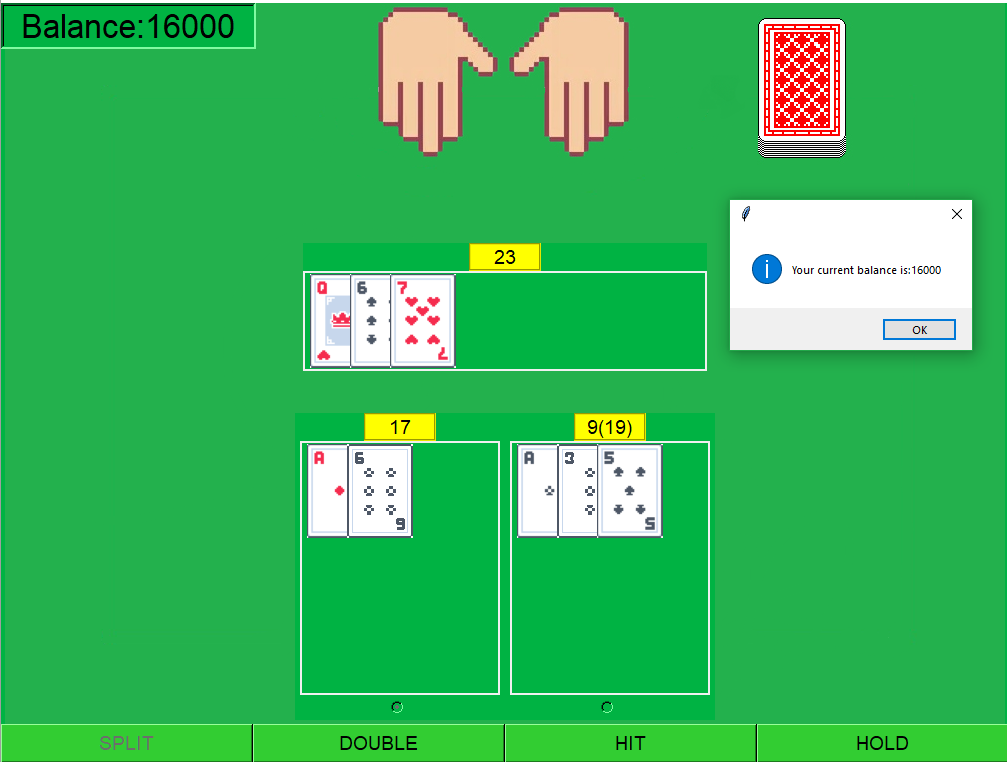


Перевіримо роботу всіх кнопок разом(натискаючи їх по черзі та комбінуючи):



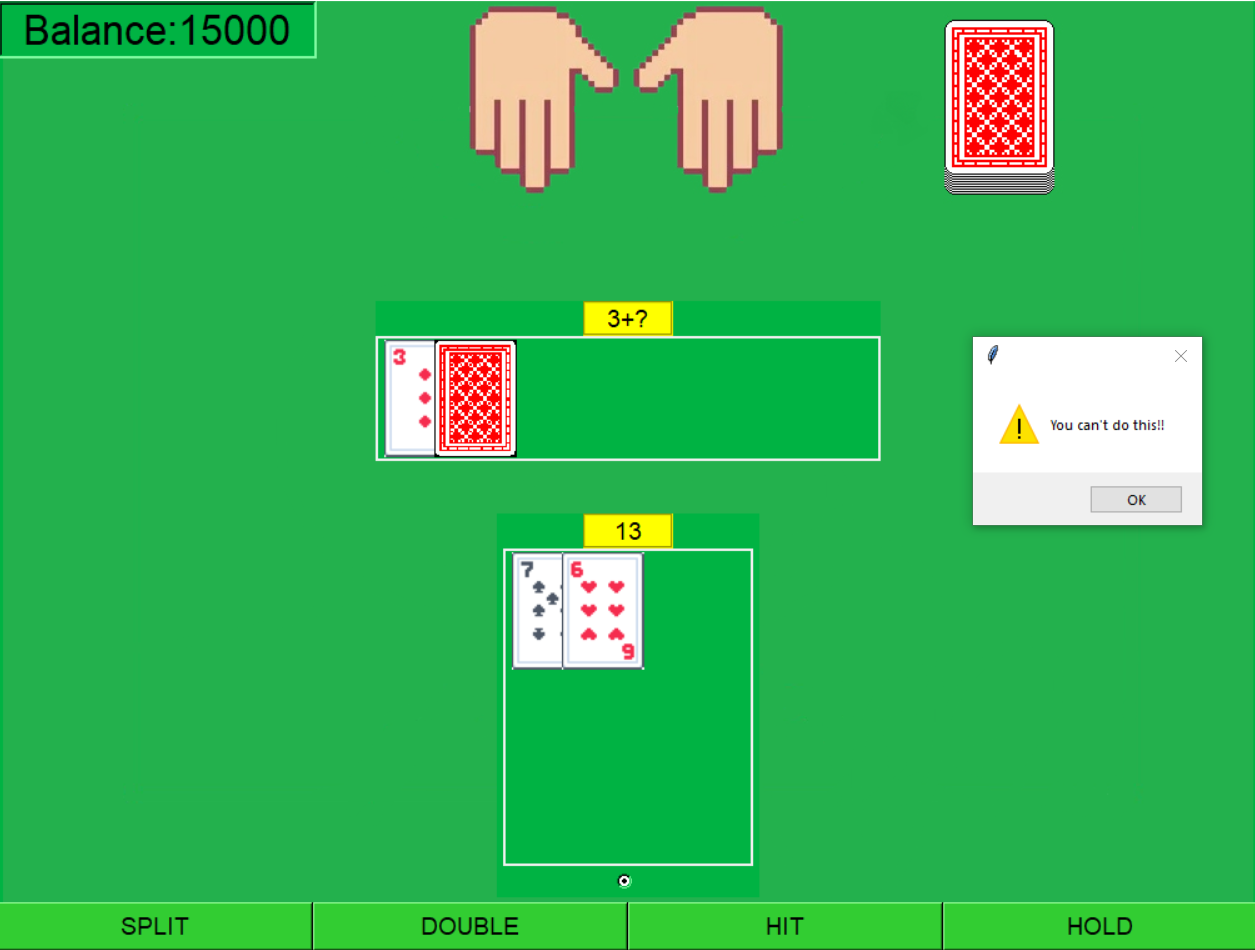




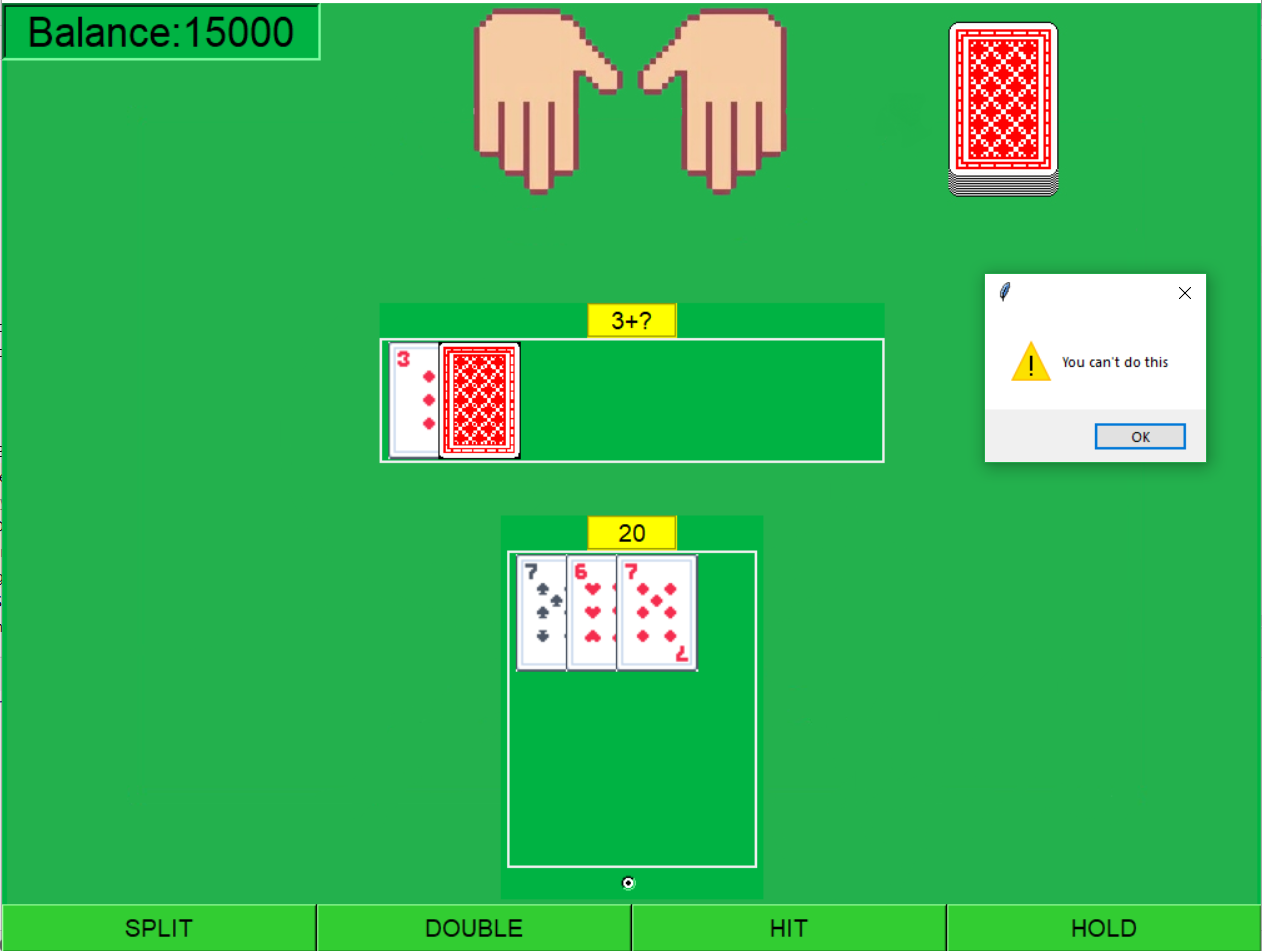


Якщо Ви не можете виконати split або duble(згідно з правилами), то Вам виведеться відповідне повідомлення:

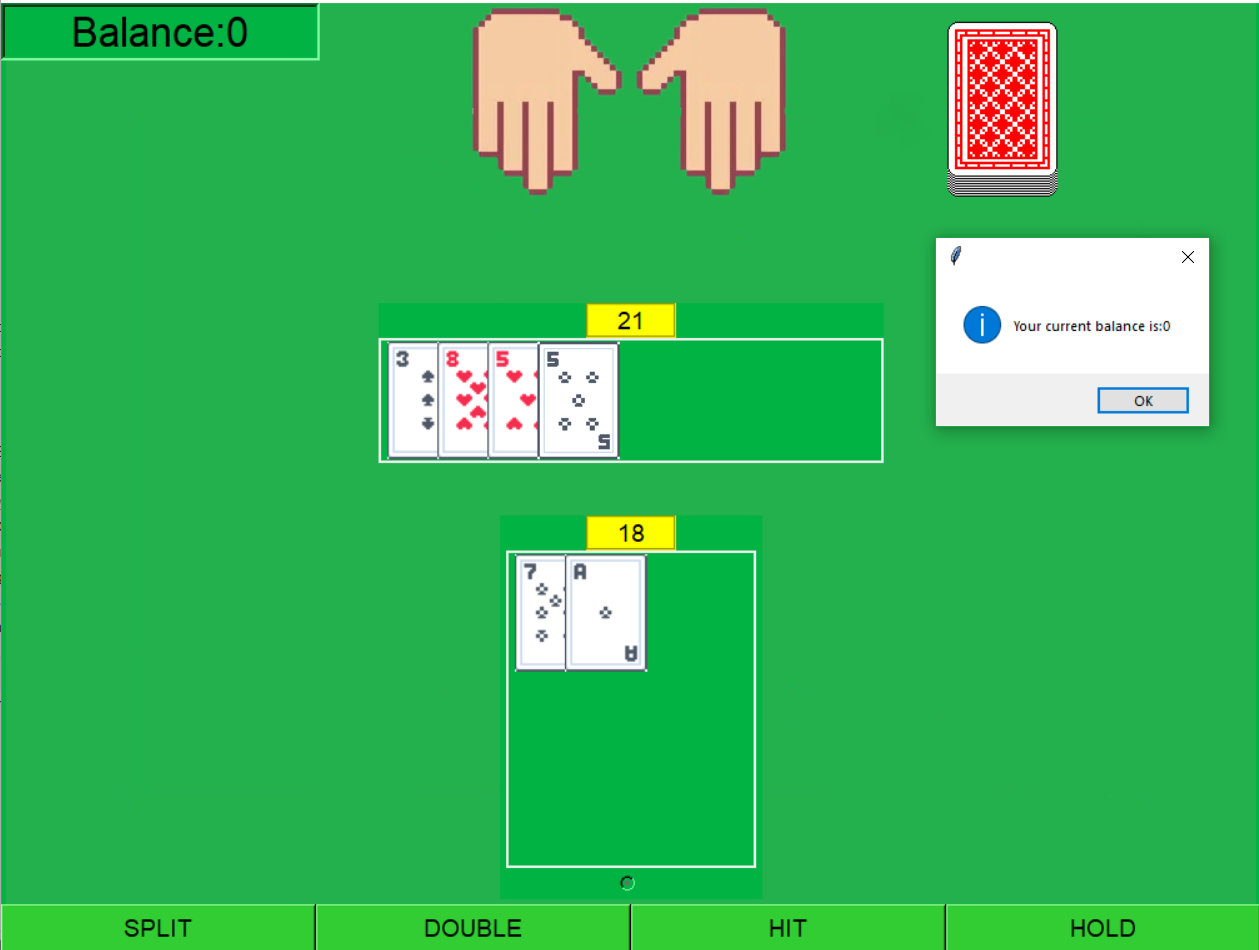
Для Split:



Для Double:



Коли баланс стає рівним нулю, то вікно з грою просто закривається



# Код програми

## BJ.py

from tkinter import PhotoImage  
import random  
  
hands\_counter = 0  
hands = []  
deck = []  
  
  
class Card:  
  
 def \_\_init\_\_(self, card\_suit, value, card\_sign):  
 self.card\_suit = card\_suit  
 self.value = value  
 self.card\_sign = card\_sign  
 self.photo = PhotoImage(file=**"images**\\**"** + self.card\_suit + **"**\\**"** + card\_sign + **".png"**)  
  
  
class Hand:  
  
 def \_\_init\_\_(self, first\_value, second\_value):  
 self.first\_value = first\_value  
 self.second\_value = second\_value  
 self.cards\_in\_hand = []  
 self.actual\_bid = 0  
 self.status = **"Active"** def count\_card\_values(self):  
 i = self.cards\_in\_hand[-1]  
 self.first\_value += i.value  
 self.second\_value += i.value  
 if i.card\_sign.\_\_eq\_\_(**"A"**):  
 temp = self.second\_value + 10  
 if temp > 21:  
 pass  
 elif temp == 21:  
 self.first\_value = self.second\_value = 21  
 else:  
 self.second\_value = temp  
 if self.first\_value < 21 < self.second\_value:  
 self.second\_value = self.first\_value  
 if self.first\_value == 21 or self.second\_value == 21:  
 self.first\_value = 21  
 self.second\_value = 21  
  
 def hit(self):  
 self.cards\_in\_hand.append(deck.pop(random.randint(0, len(deck) - 1)))  
 self.count\_card\_values()  
  
 def deal(self):  
 for i in range(2):  
 self.hit()  
  
  
def generate\_deck():  
 global deck  
 deck = []  
 card\_suits = [**"spades"**, **"clubs"**, **"hearts"**, **"diamonds"**]  
 card\_values = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 10, 10, 10]  
 card\_signs = [**"A"**, **"2"**, **"3"**, **"4"**, **"5"**, **"6"**, **"7"**, **"8"**, **"9"**, **"10"**, **"J"**, **"Q"**, **"K"**]  
 for i in card\_suits:  
 for j in range(len(card\_signs)):  
 deck.append(Card(i, card\_values[j], card\_signs[j]))  
 return deck

## Game.py

from bj import \*  
from tkinter import \*  
from tkinter import messagebox as m  
import math  
import time as t  
  
widgets = []  
hand\_number = 0  
bid = 0  
  
  
def var\_get():  
 global hand\_number  
 hand\_number = var.get()  
  
  
def var\_check():  
 temp = 0  
 for i in range(1, len(hands)):  
 if hands[i].status.\_\_eq\_\_(**"Inactive"**):  
 temp += 1  
 if temp == len(hands) - 1:  
 dealer\_play()  
 else:  
 for i in range(1, len(hands)):  
 if hands[i].status.\_\_eq\_\_(**"Active"**):  
 var.set(i)  
  
  
def check\_values():  
 temp = hands[hand\_number].first\_value  
 temp1 = hands[hand\_number].second\_value  
 if temp == 21 or temp1 == 21:  
 hands[hand\_number].status = **"Inactive"** widgets[hand\_number \* 3 + 1][**"text"**] = **"21"** hold()  
 elif temp > 21:  
 widgets[hand\_number \* 3 + 1][**"text"**] = str(temp)  
 hold()  
 if len(hands) == 2:  
 dealer\_play()  
 else:  
 var\_check()  
 elif temp != temp1:  
 widgets[hand\_number \* 3 + 1][**"text"**] = str(temp) + **"("** + str(temp1) + **")"** elif temp == temp1:  
 widgets[hand\_number \* 3 + 1][**"text"**] = str(temp)  
  
  
def start\_game():  
 global deck  
 hands.append(Hand(0, 0))  
 hands.append(Hand(0, 0))  
 hands[1].actual\_bid = bid  
 deck = generate\_deck()  
 hands[0].deal()  
 hands[1].deal()  
 show\_cards\_from\_deal()  
  
  
def show\_cards\_from\_deal():  
 global widgets  
 widgets.append(Canvas(frame\_dealer, relief=**"groove"**, width=400, height=96, bg=**"#00B343"**, borderwidth=0))  
 widgets.append(  
 Label(frame\_dealer, font=**"Arial 14"**, text=str(hands[0].cards\_in\_hand[0].value) + **"+?"**, relief=**"groove"**, width=6,  
 height=1, bg=**"yellow"**))  
 widgets.append(Canvas(frame\_player, relief=**"groove"**, width=196, height=250, bg=**"#00B343"**, borderwidth=0))  
 widgets.append(Radiobutton(frame\_player, variable=var, bg=**"#00b343"**, value=1))  
 widgets.append(Label(frame\_player, font=**"Arial 14"**, relief=**"groove"**, width=6, height=1, bg=**"yellow"**))  
 widgets[0].grid(row=1, column=0, sticky=**"S"**)  
 widgets[1].grid(row=0, column=0, sticky=**"S"**)  
 widgets[2].grid(row=1, column=0, sticky=**"N"**, padx=5)  
 widgets[3].grid(row=2, column=0, sticky=**"N"**)  
 widgets[4].grid(row=0, column=0, sticky=**"S"**)  
 widgets[0].create\_image(40, 50, image=hands[0].cards\_in\_hand[0].photo)  
 widgets[0].create\_image(80, 50, image=back)  
 for i in range(len(hands[1].cards\_in\_hand)):  
 z = math.floor(i / 4)  
 widgets[2].create\_image(40 + 40 \* (i - 4 \* z), 50 + 75 \* z, image=hands[1].cards\_in\_hand[i].photo)  
 var.set(1)  
 var\_get()  
 check\_values()  
  
  
def bid\_button():  
 global bid, balance  
 try:  
 bid = int(entry\_of\_bid.get())  
 if bid > balance or bid < 0:  
 raise ValueError  
 button\_hit[**"state"**] = ACTIVE  
 button\_split[**"state"**] = ACTIVE  
 button\_double[**"state"**] = ACTIVE  
 button\_hold[**"state"**] = ACTIVE  
 balance -= bid  
 button\_bid.destroy()  
 entry\_of\_bid.destroy()  
 balance\_label[**"text"**] = **"Balance:"** + str(balance)  
 start\_game()  
 except ValueError:  
 m.showwarning(message=**"Wrong bid"**)  
  
  
def hit\_button():  
 var\_get()  
 hands[hand\_number].hit()  
 show\_card()  
 check\_values()  
  
  
def show\_card():  
 z = math.floor((len(hands[hand\_number].cards\_in\_hand) - 1) / 4)  
 widgets[hand\_number \* 3 - 1].create\_image(40 + 40 \* (len(hands[hand\_number].cards\_in\_hand) - 1 - 4 \* z),  
 50 + 75 \* z, image=hands[hand\_number].cards\_in\_hand[-1].photo)  
  
  
def hold():  
 var\_get()  
 hands[hand\_number].status = **"Inactive"** widgets[hand\_number \* 3][**"state"**] = DISABLED  
 widgets[hand\_number \* 3 + 1][**"text"**] = hands[hand\_number].second\_value  
 var\_check()  
  
  
def split():  
 global widgets, balance, bid  
 var\_get()  
 if len(hands[hand\_number].cards\_in\_hand) == 2:  
 if hands[hand\_number].cards\_in\_hand[0].value == hands[hand\_number].cards\_in\_hand[1].value:  
 pass  
 else:  
 m.showwarning(message=**"You can't do this!!"**)  
 return  
 else:  
 m.showwarning(message=**"You can't do this!!"**)  
 return  
 if balance < hands[hand\_number].actual\_bid:  
 m.showwarning(message=**"You can't do this!!"**)  
 return  
 hands.append(Hand(0, 0))  
 hands[-1].actual\_bid = bid  
 balance -= bid  
 balance\_label[**"text"**] = **"Balance:"** + str(balance)  
 widgets[hand\_number \* 3 - 1].destroy()  
 widgets.remove(widgets[hand\_number \* 3 - 1])  
 widgets.insert(hand\_number \* 3 - 1,  
 Canvas(frame\_player, relief=**"groove"**, width=196, height=250, bg=**"#00B343"**, borderwidth=0))  
 widgets[hand\_number \* 3 - 1].grid(row=1, column=hand\_number - 1, sticky=**"N"**, padx=5)  
 widgets.append(Canvas(frame\_player, relief=**"groove"**, width=196, height=250, bg=**"#00B343"**, borderwidth=0))  
 widgets.append(Radiobutton(frame\_player, variable=var, bg=**"#00b343"**, value=len(hands) - 1))  
 widgets.append(Label(frame\_player, font=**"Arial 14"**, relief=**"groove"**, width=6, height=1, bg=**"yellow"**))  
 widgets[len(widgets) - 3].grid(row=1, column=len(hands) - 2, sticky=**"N"**, padx=5)  
 widgets[len(widgets) - 2].grid(row=2, column=len(hands) - 2, sticky=**"N"**)  
 widgets[len(widgets) - 1].grid(row=0, column=len(hands) - 2, sticky=**"S"**)  
 hands[-1].cards\_in\_hand.append(hands[hand\_number].cards\_in\_hand.pop(1))  
 hands[hand\_number].first\_value = 0  
 hands[-1].first\_value = 0  
 hands[hand\_number].second\_value = 0  
 hands[-1].second\_value = 0  
 hands[hand\_number].count\_card\_values()  
 hands[-1].count\_card\_values()  
 hands[-1].hit()  
 hands[hand\_number].hit()  
 for i in range(len(hands[hand\_number].cards\_in\_hand)):  
 z = math.floor(i / 4)  
 widgets[hand\_number \* 3 - 1].create\_image(40 + 40 \* (i - 4 \* z), 50 + 75 \* z,  
 image=hands[hand\_number].cards\_in\_hand[i].photo)  
 check\_values()  
 for i in range(len(hands[-1].cards\_in\_hand)):  
 z = math.floor(i / 4)  
 widgets[len(widgets) - 3].create\_image(40 + 40 \* (i - 4 \* z), 50 + 75 \* z,  
 image=hands[-1].cards\_in\_hand[i].photo)  
 var.set(len(hands) - 1)  
 var\_get()  
 check\_values()  
 button\_split[**"state"**] = DISABLED  
  
  
def double\_button():  
 global balance  
 var\_get()  
 if len(hands[hand\_number].cards\_in\_hand) == 2:  
 if balance < hands[hand\_number].actual\_bid:  
 m.showwarning(message=**"You can't do this"**)  
 return  
 else:  
 m.showwarning(message=**"You can't do this"**)  
 return  
 hands[hand\_number].actual\_bid += bid  
 balance -= bid  
 balance\_label[**"text"**] = **"Balance:"** + str(balance)  
  
 hands[hand\_number].hit()  
 widgets[hand\_number \* 3][**"state"**] = DISABLED  
 hands[hand\_number].status = **"Inactive"** show\_card()  
 check\_values()  
 var\_check()  
  
  
def dealer\_play():  
 global balance, frame\_player, frame\_dealer, button\_bid, button\_hit, button\_hold, button\_split, button\_double, entry\_of\_bid, widgets, hands, hand\_number, root  
 dealer\_cards\_reveal()  
  
 if hands[0].first\_value >= 17 or hands[0].first\_value == 21:  
 hands[0].second\_value = hands[0].first\_value  
 elif hands[0].second\_value >= 17 or hands[0].second\_value == 21:  
 hands[0].first\_value = hands[0].second\_value  
  
  
 else:  
 while True:  
 hands[0].hit()  
 show\_dealer\_cards()  
 temp = hands[0].first\_value  
 temp1 = hands[0].second\_value  
 if temp >= 17 or 21 >= temp1 >= 17:  
 break  
 if temp == 21 or temp1 == 21:  
 hands[0].first\_value = 21  
 hands[0].second\_value = 21  
 elif temp < 17 <= temp1:  
 hands[0].first\_value = temp1  
 elif temp >= 17 > temp1:  
 hands[0].second\_value = temp  
 widgets[1][**"text"**] = str(hands[0].first\_value)  
 for i in range(1, len(hands)):  
 if hands[i].second\_value > 21:  
 continue  
 elif hands[0].first\_value > 21:  
 balance += hands[i].actual\_bid \* 2  
 break  
 else:  
 if hands[i].second\_value == hands[0].first\_value:  
 balance += hands[i].actual\_bid  
 continue  
 elif 21 - hands[0].first\_value > 21 - hands[i].second\_value:  
 balance += hands[i].actual\_bid \* 2  
 continue  
 else:  
 pass  
 balance\_label[**"text"**] = **"Balance:"** + str(balance)  
 m.showinfo(message=**"Your current balance is:"** + str(balance))  
 t.sleep(1)  
  
 if balance == 0:  
 root.quit()  
  
 widgets = []  
 hand\_number = 0  
  
 hands = []  
 frame\_player.destroy()  
 frame\_dealer.destroy()  
 frame\_dealer = Frame(root, relief=**"groove"**, width=1000, height=100, bg=**"#00B343"**)  
 frame\_dealer.grid(row=1, column=0)  
 root.rowconfigure(0, weight=20)  
  
 frame\_player = Frame(root, relief=**"groove"**, width=196, height=250, bg=**"#00B343"**)  
 frame\_player.grid(row=2, column=0)  
 root.rowconfigure(1, weight=10)  
  
 button\_split[**"state"**] = DISABLED  
 button\_double[**"state"**] = DISABLED  
 button\_hold[**"state"**] = DISABLED  
 button\_hit[**"state"**] = DISABLED  
 button\_bid = Button(frame\_buttons, text=**"BID"**, font=**"Arial 14"**, width=22, bg=**"limegreen"**, command=bid\_button)  
 button\_bid.grid(row=1, column=3, sticky=**"WES"**)  
  
 entry\_of\_bid = Entry(frame\_buttons, font=**"Arial 25"**, width=15, relief=**"groove"**, bg=**"#00b343"**)  
 entry\_of\_bid.grid(row=0, column=3, sticky=**"WES"**)  
  
  
def dealer\_cards\_reveal():  
 global widgets  
 widgets[0].destroy()  
 widgets.remove(widgets[0])  
 widgets.insert(0, Canvas(frame\_dealer, relief=**"groove"**, width=400, height=96, bg=**"#00B343"**, borderwidth=0))  
 widgets[0].grid(row=1, column=0, sticky=**"S"**)  
 widgets[0].create\_image(40, 50, image=hands[0].cards\_in\_hand[0].photo)  
 widgets[0].create\_image(80, 50, image=hands[0].cards\_in\_hand[1].photo)  
  
  
def show\_dealer\_cards():  
 temp = len(hands[0].cards\_in\_hand)  
 widgets[0].create\_image(40 + 40 \* (temp - 1), 50, image=hands[0].cards\_in\_hand[-1].photo)  
  
  
balance = 25000  
root = Tk()  
root.geometry(**"1007x760"**)  
root.resizable(height=False, width=False)  
root.configure(bg=**"#00b343"**)  
root.title(**"BlackJack"**)  
background = PhotoImage(file=**r"images/background.png"**)  
backg = Label(root, width=1000, height=750, image=background, bg=**"#00b343"**)  
backg.grid(row=0, column=0, rowspan=3)  
  
var = IntVar()  
var.set(1)  
  
balance\_label = Label(root, bg=**"#00b343"**, font=**"Arial 25"**, width=13, text=**"Balance:"** + str(balance), relief=**"sunken"**,  
 borderwidth=3)  
balance\_label.grid(row=0, column=0, sticky=**"WN"**)  
  
frame\_dealer = Frame(root, relief=**"groove"**, width=1000, height=100, bg=**"#00B343"**)  
frame\_dealer.grid(row=1, column=0)  
root.rowconfigure(0, weight=20)  
  
frame\_player = Frame(root, relief=**"groove"**, width=196, height=250, bg=**"#00B343"**)  
frame\_player.grid(row=2, column=0)  
root.rowconfigure(1, weight=10)  
  
frame\_buttons = Frame(root, relief=**"groove"**, width=1000, height=100, bg=**"#00B343"**)  
frame\_buttons.grid(row=3, column=0, sticky=**"WES"**)  
root.rowconfigure(2, weight=1)  
  
button\_split = Button(frame\_buttons, text=**"SPLIT"**, font=**"Arial 14"**, width=22, bg=**"limegreen"**, state=DISABLED,  
 command=split)  
button\_split.grid(row=1, column=0, sticky=**"WES"**)  
  
button\_double = Button(frame\_buttons, text=**"DOUBLE"**, font=**"Arial 14"**, width=22, bg=**"limegreen"**, state=DISABLED,  
 command=double\_button)  
button\_double.grid(row=1, column=1, sticky=**"WES"**)  
  
button\_hit = Button(frame\_buttons, text=**"HIT"**, font=**"Arial 14"**, width=22, bg=**"limegreen"**, state=DISABLED,  
 command=hit\_button)  
button\_hit.grid(row=1, column=2, sticky=**"S"**)  
  
button\_hold = Button(frame\_buttons, text=**"HOLD"**, font=**"Arial 14"**, width=22, bg=**"limegreen"**, state=DISABLED,  
 command=hold)  
button\_hold.grid(row=1, column=3, sticky=**"WES"**)  
  
button\_bid = Button(frame\_buttons, text=**"BID"**, font=**"Arial 14"**, width=22, bg=**"limegreen"**, command=bid\_button)  
button\_bid.grid(row=1, column=3, sticky=**"WES"**)  
  
entry\_of\_bid = Entry(frame\_buttons, font=**"Arial 25"**, width=15, relief=**"groove"**, bg=**"#00b343"**)  
entry\_of\_bid.grid(row=0, column=3, sticky=**"WES"**)  
  
back = PhotoImage(file=**"images**\\**card\_back.png"**)  
root.mainloop()

# Список використаних джерел:

<https://metanit.com/> - Metanit

<https://chat.openai.com/> - Chat GPT

<https://docs.python.org/> - документація мови Python

<https://docs.python.org/uk/3/library/tkinter.html> - документація модуля ‘Tkinter’

Асети для карт і всі елементи дизайну взяті з відкритих джерел та намальовані власноруч  
  
Дякуємо!!!!!