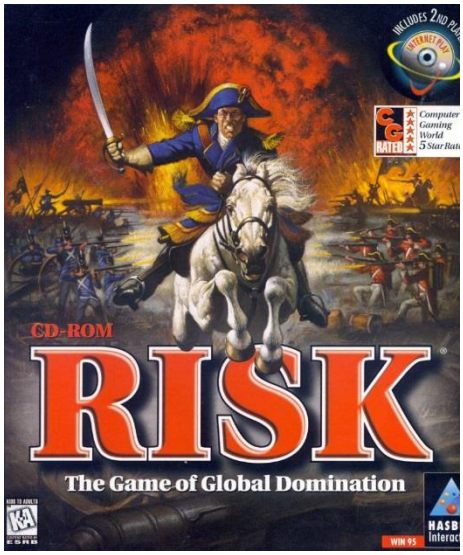
עבודה ב OOP 31631 סמסטר א' התשע"ט,
עבודה מסכמת

משחק- סיכון RISK-risk the game of global domination



תאריך ההגשה: 25/02/2019

שמות הסטודנטים:

אורי שדה, ת.ז. 318262128 orisade12@gmail.com

גל אורן, ת.ז. 206232506 gal287199@gmail.com

בהנחיית המרצה: יעל ארז

ציון:

-

תאור הספריה :

הספרייה risk מנהלת משחק risk ומספקת מחלקות ומתודות המתאימות למשחק.

- **הספרייה מאפשרת :**
- **אפשרות לבחור את מספר השחקנים בין 2-4.**
- **אפשרות לקבוע את סדר התורות ע"י הטלת קובייה.**
- **יצירה של מפה של 49 ארצות ו6 יבשות**
- **חלוקה התחלתית של חיילים לשחקנים על מנת שיאכלסו את המדינות- משתנה לפי מספר השחקנים.**
- **ניהול סבב כיבושים ראשוני ע"י השחקנים עד שהמפה מתמלאת וסבב נוסף לחיזוק השליטה ע"י השחקנים.**
- **ניהול שלושת השלבים שמתבצעים במהלך המשחק draft,attack,reinforcment .**
- **חלוקת קלפי כיבושים לשחקנים ומתן חיילי בונוס בהתאם לשחקנים.**
- **מאפשרת שמירה בכל שלב ושחזור המשחק לכל נקודה .**
- **הצגת סטטיסטיקה על המשחקים-השחקן המצטיין השחקן עם הכי הרבה הפסדים – השחקן ששיחק הכי הרבה זמן ,השחקן ששיחק הכי הרבה משחקים ,זמן ממוצע של משחק בשניות**
- *** הספרייה מלווה בליווי של ממשק גרפי ידודתי המציג את הנתונים שמנוהלים על ידי הספריה כמו כן ישנו שיר רקע המלווה את השיר וצלילם לפפעולות שונות במשחק(כמו כיבוש ארץ).**
- **שימוש בנושאים אשר נלמדו בקורס Encapsulation, Inheritance, Templates, Operator overloading, STL**

המחלקות שמומשו במסגרת הספרייה :

Stage–מחלקה אבסטרקטית לשלב משחק

Place map–מחלקה לשלב המשחק הראשוני-מיקום החיילים עד שהמפה מלאה

Place troops–מחלקה שאחראית על ניהול השלב השני באתחול המפה- עד תום מיקום החיילים על פני המפה

Draft–מחלקה זו אחראית על השלב הראשון בסבב המשחק .

Attack– מחלקה זו אחראית על שלב התקיפה בסבב המשחק

Reinforcement– מחלקה זו אחראית על השלב השלישי בסבב משחק של שחקן.

Risk–המחלקה הראשית בה מתנהל המשחק ו הgui הראשי

Player–מחלקה המממשת שחקן- שומרת נתונים עבורו

Card – מחלקה המממשת קלף משחק

Land–מחלקה המממשת יבשת

Country– מחלקה המממשת מדינה.

Menu-מחלקה המנהלת את ה gui של תפריט המשחק

Vector- מחלקה המנהלת מבנה נתונים של וקטור –templat .

Playerstas- class templet המנהלת את מבנה הנתונים של הסטטיסטיקות של השחקנים .

Playerstatwrap- מחלקה ששומרת את המשתנים לגבי סטטיסטיקות של שחקן.

Statsguimain-מחלקה המנהלת את הguiשל הסטטיסטיקה.

המחלקה cube :

המחלקה מנהלת אובייקט של קוביה –מגרילה מספרים בין 1 ל6 - עבור השלב הראשוני של קביעת התור ממיינת את התוצאות ועבור המשך משחק משמשת לקרבות בשלב הattack.

```
bool operator>(const Cube&) const;
bool operator==(const Cube&) const;
```

Ctor	Cube(); (const Player&);
אופרטור =	bool operator<(const Cube&) const; bool operator>(const Cube&) const; bool operator==(const Cube&) const;
מציגה ומסתירה את הקובייה על פני ה gui.	void hide(); void show();

השמה והוספה של חיילים עבור כיבוש של מדינות וטריטוריות.	void choosePic(int);
	void setValue(int);
	int getValue() const;void setBonusTroops(const int);

קביעת הנתיב של התמונה שאותה רוצים לטעון לפי ההגרלה.	Void setpath(std::string)
שומרת את השם של השחקן צבע סדר ו id . טוענת את הפרמטרים של השחקן שנשמרו לקובץ .	void save(std::ofstream& file) void load(std::ifstrsceam& file)
מציגה את הקלפים של השחקן על גבי ה gui.	void show()
(-מנקה את הקלפים של השחקן במידה והשתמש בהם – מה gui.	void cleanCards(std::vector<Card> cards)
	Members
מכיל פרטים אודות השחקן בהתאמה ./	int id name color]
מחזיק את הסדר של השחקן בתור	int order;
מחזיק את הקלפים של השחקן.	std::vector<Card> cards;
מחזיק את החיילים של השחקן שמגיעים לו בהתאם לכיבישו והיבשות בהם הוא שולט.	int bonusTroops;

המחלקה Player :

Ctor	`Player(const std::string name = "default", const std::string color = "None", int order = 0, int id = -1);` `Player(const Player&);`
אופרטור =	`Player& operator=(const Player&);`
מנקה את הקלפים של השחקן במידה והשתמש בהם – על גבי ה gui.	`void cleanCards(std::vector<Card> - cards)`
השמה והוספה של חיילים עבור כיבוש של מדינות וטריטוריות.	`void addBonusTroops(const int);` `void setBonusTroops(const int);`
מוסיפה קלף לשחקן	`Card addCard();`
שומרת את השם של השחקן צבע סדר ו id . טוענת את הפרמטרים של השחקן שנשמרו לקובץ .	`void save(std::ofstream& file)` `void load(std::ifstrsceam& file)–`
מציגה את הקלפים של השחקן על גבי ה gui.	`void show()–`
(-מנקה את הקלפים של השחקן במידה והשתמש בהם – על גבי ה gui.	`void cleanCards(std::vector<Card> cards)`
	`Members`
מכיל פרטים אודות השחקן בהתאמה ./	int id name color|
מחזיק את הסדר של השחקן בתור	`int order;`
מחזיק את הקלפים של השחקן.	`std::vector<Card> cards;`
מחזיק את החיילים של השחקן שמגיעים לו בהתאם לכיבישו והיבשות בהם הוא שולט.	`int bonusTroops;`

המחלקה Card: *כשמה מממשת קלף שחקן עבור המשתמש –בעת שכובש טריטוריה מקבל קלף- השווי של הקלף בחיילים מתנהל לפי סדרת פיבונאצי.*

```
        Card& operator=(const Card&);
        bool operator==(const Card&) const;
bool operator!=(const Card&) const;
```

מקדמת את הערך של מסחר ב3 קלפים לפי סדרת פיבונאצי.	`static void nextBonus();`
מחזירה את הערך הנוכחי של מספר ב3 קלפים	`static int  getBonus();`
ctors	`Card(cardType cardType = cardType::Default);` `Card(const Card&);`
סוגי הקלפים הקיימים – כדי לבצע החלפה צריך 3 זוגות זהים או 3 שונים כאשר ג'וקר מתאים לכולם .	`enum cardType { Joker, Solider, Cannon, Horse, Default };`
מבצע את ההגרלה של הקלפים מבין הסוגים הקיימים	`void setRand();`
	`void setButton(QPushButton*);`
קישור בין קלף לבין לחצן ב Gui שמתאים לו .	`void setButton(QPushButton*);`
עדכון של קלפי המשחק של השחקן בgui .	`void Card::update()`
שמירה וטעינה של סוג הקלף לקובץ.	`void save(std::ofstream& file) const;`

	void load(std::ifstream& file);
מנקה את התצוגה של הקלף מהגui.	void clean();
מחזיר את סוג הקלף	int getType() const;
מתודה המנהלת את הלחיצה של המתמש על פני הקלף- במידה וקיים קלף בידי השחקן אם הוא לוחץ אליו מציגה אותו עם צל ולחיצה נוספת מבטלת את הצל. syntax Slots- של gui שמשמעותו-זו המתודה שמטפלת ב event בעת לחיצה –במקרה הזה המתודה מתקרת ללחצן שמקושר לקלף שעל גביו מוצגת תמונה של קלף.	public slots:- void click();
	Members
שלושת הערכים האחרונים בשווי הקלף לפי כמות חיילים – כאמור החלטנו שהשווי של מסחר ב3 קלפים ילך ויתקדם לפי סדרת פיבונאצי .	static int bonus1; static int bonus2; static int bonus3;

STAGE המחלקה – מחלקה אבסטרקטית המכילה מתודות הדרושות ליישום שלבי המשחק.

מאתחל את הדגלים nexten וenteren אשר מתפעלים את לחצן ה enter שתפקידו לבצע אישור של פעולות בשלבי המשחק וכפתור ה next שמאפשר לעבור בין שלבי המשחק .	Stage(Ui::my_riskClass&, Player& player, Move& moveUi);
מבצע פעולות המקדימות את השלב כמו חישבוים (חישוב מספר החיילים המגיע למשל ב draft) והצגה על פני ה gui ואתחול משתנים	virtual void preAction()=0
ביצוע כל הפעולות הקשורות בשלב אותו מבצעים	virtual void action()=0;
מבצע בדיקות האם מתקיימים התנאים שעבורם ניתן לעבור לשלב הבא ת מציג נתונים על גבי ה Gui .	virtual void afterAction()=0;
מציג את מספר החיילים שאותם זכאי המשתמש למקם על גבי המדינות שבבעלותו וכמו כן מספר החיילים שאיתם הוא בוחר לתקוף בשלב התקיפה .	virtual void showTroops();
מאפשר שליטה על הטקסט שמוצג ב showtroops – הוספת חיילים למקום מסוים- תקיפה עם יותר חיילים .	virtual void plus(int sel = 0);
בדומה לvoid plus רק שמאפשר להפחית .	virtual void minus(int sel = 0);
Getters למצב הלחצנים.	bool getNexEn(); bool getEnterEn();
ettersד ללחצנים. .	void setNextEn(const bool en); void setEnterEn(const bool en);
Setter שמבצע השמה לרפרנס של השחקן הנוכחי שמשחק .	void setPlayer(Player&);
מחזיר את מספר חיילי התגבור שלהם זכאי השחקן- שימושי בשלב ה draft .	int getBonusTroops(int sel = 0);
בודק אם מדינה שנלחצה שייך לשחקן הנוכחי שמשחק – על סמך כך נקבעות החלטות בשלבים שונים במשחק	bool isMine(int sel = 0);
	Members
רפרנס לשחקן הנוכחי שמשחק	Player& player;
מצב כפתור ה next	bool nextEn;
מצב כפתור ה enter	bool EnterEn;

המחלקה PlaceMap : יורשת מstage

אחראית למימוש השלב הראשון של כיבוש החלקים במפה ע"י המשתתפים.

void preAction()	אחראית להצגה של נתונים של השלב הראשון על גבי ה gui.
void action()	אחראית למימוש השלב הראשון.
void afterAction()	מבצעת השמה לנתונים על פני הgui לאחר ביצוע השלב.
Members	
int& counter;	משמש לספור כמה חיילים חילקנו עד עכשיו .

המחלקה PlaceTroops : יורשת מstage

אחראית לביצוע השלב השני באתחול המשחק- מיקום כל החיילים שחולקו לשחקנים בשלב הראשוני על פני הלוח.

void action()	אחראית למימוש השלב השני.
void preAction()	אחראית להצגה של נתונים של השלב השני על גבי ה gui.
void afterAction()	-
void showInfo();	מציג מידע על על גבי ה gui.
Members	
int& counter;	משמש לספור כמה חיילים חילקנו עד עכשיו .

המחלקה draft : יורשת מstage-מממשת את שלב ה draft –בשלב זה השחקן ממקמם את החיילים שהוא מקבל בתחילת כל תור במדינות שבבעלתו – מספר המדינות נקבע על סמך הכיבושים שלו ומספר היבשות שהוא כובש.

void action()	אחראית למימוש השלב של הdraft
void preAction()	אחראית להצגה של נתונים של השלב הdraft על גבי ה gui ובנוסף לחישוב מספר החיילים המגיע לשחקן בתחילת התור והצגת הקלפים שבבעלותו.
void afterAction()	מציג מידע על על גבי ה gui.
voidshowBonusTroops(std::string)	מציג מידע על על גבי ה gui.
Land::landInfo calcTroops();	מחשבת את מספר החיילים המגיעים לשחקן בהתאם לכיבשיו והקלפים שבידיו- במידה ובידיו 5 קלפים חייב לסחור בהם .
void plus(int sel = 0);	מנהלת את הלחיצות על כפתור ה+ למיקום חיילם בטריטוריה מסויימת.
void minus(int sel = 0);	מנהלת את הלחיצות על כפתור ה- למיקום חיילם בטריטוריה מסויימת.
void convert();	אחראית להחלפת קלפים של שחקן תמורת חיילים לתגבור
members	
std::vector<Card>& cards;	רפרנס לקלפים של הgui.
std::vector<Land*> lands;	פויינטר ליבשות של המשחק.

המחלקה attack : יורשת מstage -אחראית על מימוש שלב ההתקפה של שחקן – בשלב זה השחקן יכול לכבוש מדינות הסמוכות לטריטריות שבבעלתו זאת ע"י לחיצה על מדינה שלו ומדינה שכנה והתמודדות בקרב של הטלת קוביות- השחקן יכול להטיל עד 3 המותקף יכול להגן עד 2.

אחראית למימוש השלב של הattack	`void action()`
אחראית להצגה של נתונים של השלב הdraft על גבי ה gui ובנוסף לחישוב מספר החיילים המגיע לשחקן בתחילת התור והצגת הקלפים שבבעלותו.	`void preAction()`
מציג מידע על על גבי ה gui.	`void afterAction()`
מציג מידע על על גבי ה gui.	`void plus(int sel = 0)`
מחשבת את מספר החיילים המגיעים לשחקן בהתאם לכיבשיו והקלפים שבידיו- במידה ובידיו 5 קלפים חייב לסחור בהם .	`void minus(int sel = 0)`
	`Members`
מסמל אם מגיע card חדש או לא	`int newCard;`
פוינטר לקלף במקרה של כיבוש מדינה.	`Card*  card;`
פוינטר לקובץ אודיו שעושה צליל של פיצוף כאשר יש כיבוש של מדינה .	`QMediaPlayer* boom;`

המחלקה reinforcement : יורשת מstage-אחראית למימוש שלב התגבור בו המשתמש יכול להעביר חיילים מטריטוריה מסוימת שלו לטריטוריה אחרת שלו בתנאי שיש מסלול רציף שמחבר אותם.
בדומה לשלבי המשחק הקודמים ממש את המתודות:

```
void action();
void preAction();
void afterAction();
```

המחלקה land –מממשת יבשת של המשחק במשחק יש 6 יבשות

אובייקט המחזיר מידע לגבי הבונוס המגיע למי שמחזיק ביבשת וstring להצגה על גבי ה gui – במידה ושחקן כבש את כל היבשת מציג- כמה שווה היבשת והסיבות שבגינן הוא זכאי למספר החיילים שקיגבל.	`struct landInfo {`
	`int bonus;`
	`std::string reason;`
	`};`

השמה של יבשת ליבשת .	`Land& operator=(Land&)`
מאתחל land בשם וbandinfo	`Land(std::vector<std::string>, Land::landInfo)`
מחשבת את מספר החיילים אשר להם זכאי שחקן עבור שליטה במדינות ביבשת.	`Land::landInfo calcTroops(Player&)`
שמירה וטעינה של פרמטרים הקשורים למחלקה לקובץ –ריצה בלולאה על כל המדינות ששיכות ליבשת ושימוש בפונקציית השמירה והטעינה של מדינה .	`void save(std::ofstream& file);` `void load(std::ifstream& file, std::map<std::string, Player>& playersMap);`
	`Members`
מחזיק את המדינות הנמצאות על היבשת.	`std::vector <Country> countries;`

המחלקה country : יורשת מ QMainWindow על מנת ליצור בה כפתור בgui שיתקשר למדינה.

קונסטרקטור וקופי קונסקרטור	`//ctor`
	`Country(const std::string,`
	`  Player player = Player())`
	`Country(const Country &);`
	`Country& operator=(Country&);`
	`Country();`

Setters למשתני המחלקה –שכנים, כפתור שאליו המדינה משוכית, פוינטר לגui,שחקן שולט.	`//setters`
	`void setOwner(const Player&);`
	`static void setUI(Ui::my_riskClass*);`
	`void setButton(QPushButton*);`
	`void`
	`setNehibors(std::vector<Country*>);`

getters למשתני המחלקה	`//getters`
	`Player getOwner() const;`
	`std::string getName() const;`
	`int getTroops();`

מתודה לבדיקה אם מדינה סמוכה- על מנת לדעת אם ניתן להעביר אליה חיילים או לתקוף אותה בשלבי המשחק השונים-בודק קרבה מסדר ראשון.	`bool isNear(std::string) const;`

בודק האם קיים מסלול רציף של מדינות של המשתמש כלומר קרבה מסדר גבוה מאחד – **באמצעות backtracking**	`bool Country::validPath(const`
	`std::string dest,`
	`std::map<std::string,bool>& seen)`

שמירה וטעינה של המשחק.	`void save(std::ofstream& file) const;` `void load(std::ifstream& file);`
מעדכן את שתי המדינות האחרונות שנלחצו chosen ו chosen2 וכמו כן מדפיס נתונים לגבי השייכות שלהם למשתמשים או אם הם פנויות.	`public slots:` `void init();`
	`Members:`
מצביעם לשתי המדינות האחרונות שנלחצו – שימושי בכל שלבי המשחק שרוצים להעביר מידע(חיילים) בין שתי מדינות.	`static Country* chosen;` `static Country* chosen2;`
פוינטר לגui הראשי.	`static Ui::my_riskClass* ui;`
מספר החיילים שעל גבי המדינה	`int troops;`
השם של המדינה	`std::string name;`
השכנים של המדינה	`std::vector<Country*> nehibors;`
הכפתור המשויך למדינה	`QPushButton* button;`
הבעלים של המדינה –נדרש רק לדעת פרמטרים שלו לכן עותק	`Player owner;`

המחלקה move :

סטרים לאתחול מספר מינימלי ומקסימלי של שחקנים להעברה	`void setMax(int);` `void setMin(int);`
קבלת מספר חיילים להעברה	`int getBonusTroops(int sel = 0);`
אתחול יעד ומקור להעברת החיילים	`void setFrom(Country*);` `void setTo(Country*);`
	`Members`
מתודות ללחצנים של פלוס ומינוס לקביעת מספר החיילים	`void plus(int sel = 0);` `void minus(int sel = 0);`
מתודת אישור	`void end();`
ממשק משתמש אבא	`QWidget* father;`
ממשק משתמש שלו.	`Ui::Move ui;`

המחלקה menu : יורשת מ `QWidget` על מנת ליצור בה כפתור בגui שיתקשר לתפריט.

קונסטרקטור וקופי קונסקרטור	`//ctor` `menu(QWidget * parent = Q_NULLPTR);`
מתודת לחצן לאישור הנתונים ותחילת משחק	`void end();`
טיפול בבחירת מספר שחקנים	`void spin(int);`
הטלת קוביות	`void cube();`
שמירה וטעינה של המשחק.	`void save(std::ofstream& file) const;` `void load(std::ifstream& file);`
טעינת קובץ משחק באמצעות dialog .	`void LOAD();`
טעינת קובץ סטטיסטיקות ופתיחת ממשק משתמש לסטטיסטיקות	`void loadStats();`
טעינת קובץ ברירת מחדל לסטטיסטיקות מתעדכן באופן רציף בכל משחק .	`void defaultStats();`
	`Members:`
ממשתק משתמש.	`Ui::menu ui;`
אוביקט המשחק.	`risk w;`
ממשק משתמש סטטיסטיקות.	`statsGuiMain s;`
קוביות לקביעת סדר	`Cube cubes[4];`
שחקנים לאיתחול	`Player players[4];`
היסטוגרמה למיון השחקנים על סמך הטלת הקוביות- עמודות זה מספרים מ1-6 שורות –הפוינטרים לשחקנים.	`Player* hist[6][4];`
מספר השחקנים הנבחר	`int numberOfPlayers;`
דגל – מציין אם נעשתה הטלה-חייב לבצע הטלה על מנת להתחיל את המשחק	`bool toss;`
דגל מצב ברירת מחדל	`bool defaultStats_;`

המחלקה risk : יורשת מ QMainWindow על מנת ליצור בה כפתור בגui שיתקשר למשחק.

קונסטרקטור	//ctor risk(int numberOfPlayers, QWidget *parent = 0);
מספר שחקנים	char get_number_of_player();
מספר שחקנים	void set_number_of_player(char num);
הטלת קוביות	void cube();
שמירה וטעינה של המשחק.	void save(std::ofstream& file) const; void load(std::ifstream& file);
קידום שחקן	void incPlayer();
איתחול חיילים לשחקנים	void initTroops();
הצגת נתוני החיילים במסך	void showTroops();

פעולות על הקונטיינר queue	bool empty() const;
	void pop();
	void push(const Player&);

טיפול בשמירת וטעינת משחק – מתודות לגוי ופינמיות	void save(std::ofstream& file);
	void load(std::ifstream& file);
	
	void LOAD();
	void SAVE();

טעינת סטטיסטיקות עם ממשק משתמש	PlayerStats* loadStats(std::string fileName = "Resources/stats.csv"); void loadStatsGui();
הצגת נתונים מחדש על המסך (refresh)	void showInfo(); void updateStage();
מתודת לחיצת לחצן של קל,	void cardClick(int sel = 0);
פתיחת גוי להצגת הודעה	void showMess(std::string);
אישור פעולה	void ENTER();
שמירת משחק + סטטיסטיקה	void SAVE_ALL();
שמירת סטטיסטיקה	void saveStats();
יציאה מהמשחק ושמירת סטטיסטקות	void EXIT();
מעבר מצב משחק	void nextState();
	Members:
ממשתק משתמש.	Ui::my_riskClass ui;
תור שחקנים	std::queue<Player> queue;
יבשות	Land SouthAmerica, NorthAmerica, Europe, Asia, Australia, Africa;
כפתורים לכל יבשת -> ממשק משתמש	std::vector<QPushButton*> SA_buttons; std::vector<QPushButton*> NA_buttons; std::vector<QPushButton*> EU_buttons; std::vector<QPushButton*> AS_buttons; std::vector<QPushButton*> AU_buttons; std::vector<QPushButton*> AF_buttons;
מספר שחקנים	char numberOfPlayers;
מספר מדינות	int numberOfCountries;

מספר חיילים לכל שחקן	int troopsPerPlayer;
מונה מספר חיילים שנשלחו-משמש בשני השלבים הראשונים במשחק מציג כמה נשאר עוד למקם על פני ה Gui.	int counter;
דגל קלף חדש	bool newCard;
מחרוזת מצב משחק נוכחי	std::string stage;
קלפים שמיוחסים לממש משתמש	std::vector<Card> cards;
מתודת משחק משתמש ללחיצת כרטיס	void cardClick(int sel = 0);
אובייקט סטטיסטיקות	PlayerStats stats;
זמן תחילת משחק- לשמירה בסטטיסטקות.	time_t startTime;
אוביקט ממשק משתמש של הודעה	Mess mess;
ממפה מחרוזת של שחקן לשחקן	std::map<std::string, Player> playersMap;
אוביקט ממשק משתמש להעברת שחקנים בין מדינות	Move moveUi;
פוינטר לאוביקט מצב נוכחי	Stage* currentStage;
ממפה מחרוזת של מצב למחרוזת של מצב שבא לפניה- לצורך אתחול נתונים.	std::map<std::string, std::string> stage2preStage;

המחלקה PlayerStats: מכילה מטודות כלליות של סטטיסטקות על השחקנים – המידע על ה שחקנים שמור ב container templet מסוג vector עליו יפורט בהמשך. הconteiner מכיל אובייקטים מסוג PlayerStatWrap המכיל מידע סטטיסטי על כל השחקנים שאי פעם שיחקו.

קונסטרקטור	//ctor
	PlayerStats();

גטרים – קבלת מספר מקסימלי של : זמן משחק,נצחונות,הפסדים,מספר משחקים	PlayerStats::Info<int> getMaxWins(); PlayerStats::Info<int> getMaxLooses(); PlayerStats::Info<int> getMaxTimePlayed(); PlayerStats::Info<int> getMaxGamesPlayed();
מחרוזת של זמן ממוצע של משחק	std::string getAvgTime();
טעינה ושמירה של סטטיסטיקות	void loadStats(std::ifstream& file); void saveStats(std::ofstream& file);
שמירה וטעינה של המשחק.	void save(std::ofstream& file) const; void load(std::ifstream& file);
הוספת סטטיסטיקות של שחקן על המעטפת שלו	void addToStats(const PlayerStatWrap&);
ניקוי סטטיסטיקות	void clear();
גודל של מספר שחקנים ששמורים בסטטיסטיקה	int size();
	Members:
אוביקט של הקונטיינר האישי שלנו Vector	Vector<PlayerStatWrap> players;

המחלקה PlayerStatWrap: המחלקה מכילה מידע סטטיסטי על שחקנים במשחק .

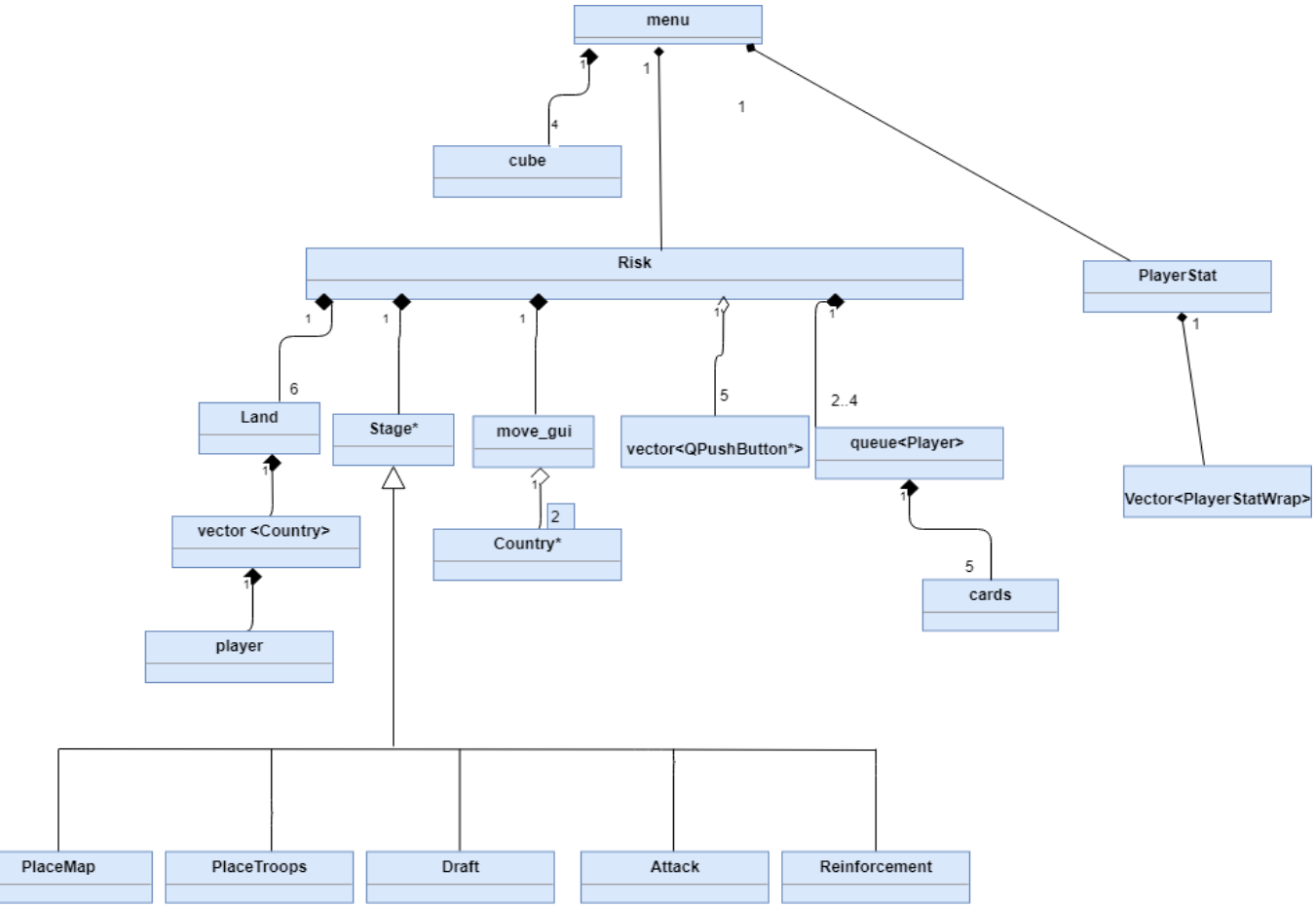
קונסטרקטורים	//ctor PlayerStatWrap(const PlayerStatWrap& src); PlayerStatWrap(const Player& player,int win, int elapsed_seconds);PlayerStatWrap();;
קלאס זה חבר של קלאס הסטטיסטיקה של השחקן – הם באים ביחד.	friend class PlayerStats;
שמירה וטעינה של סטטיסטיקות האוביקט לקובץ ומקובץ	void loadStats(std::ifstream& file); void saveStats(std::ofstream& file);
אופרטורים	PlayerStatWrap& operator+(const PlayerStatWrap& src); PlayerStatWrap& operator=(const PlayerStatWrap& src);
	Members:
זמן ששיחק	int timePlayed;
מספר משחקים ששיחק	int gamesPlayed;
מספר נצחונות	int wins;
מספר הפסדים	int looses;
תז	int id;
שם	std::string name;

המחלקה Vector : מממשת את מבנה הנתונים וקטור בו אנו משתמשים לשמור מידע סטטיסטי על המשתמשים-מחלקה template. בחרנו לממש מבנה זה כיוון שמספר השחקנים הנטען אינו יודע ווקטור הוא מבנה נוח בוא אפשר להוסיף באופן דינמי שאינו ידוע מראש כמות פריטים מבוקשת. \

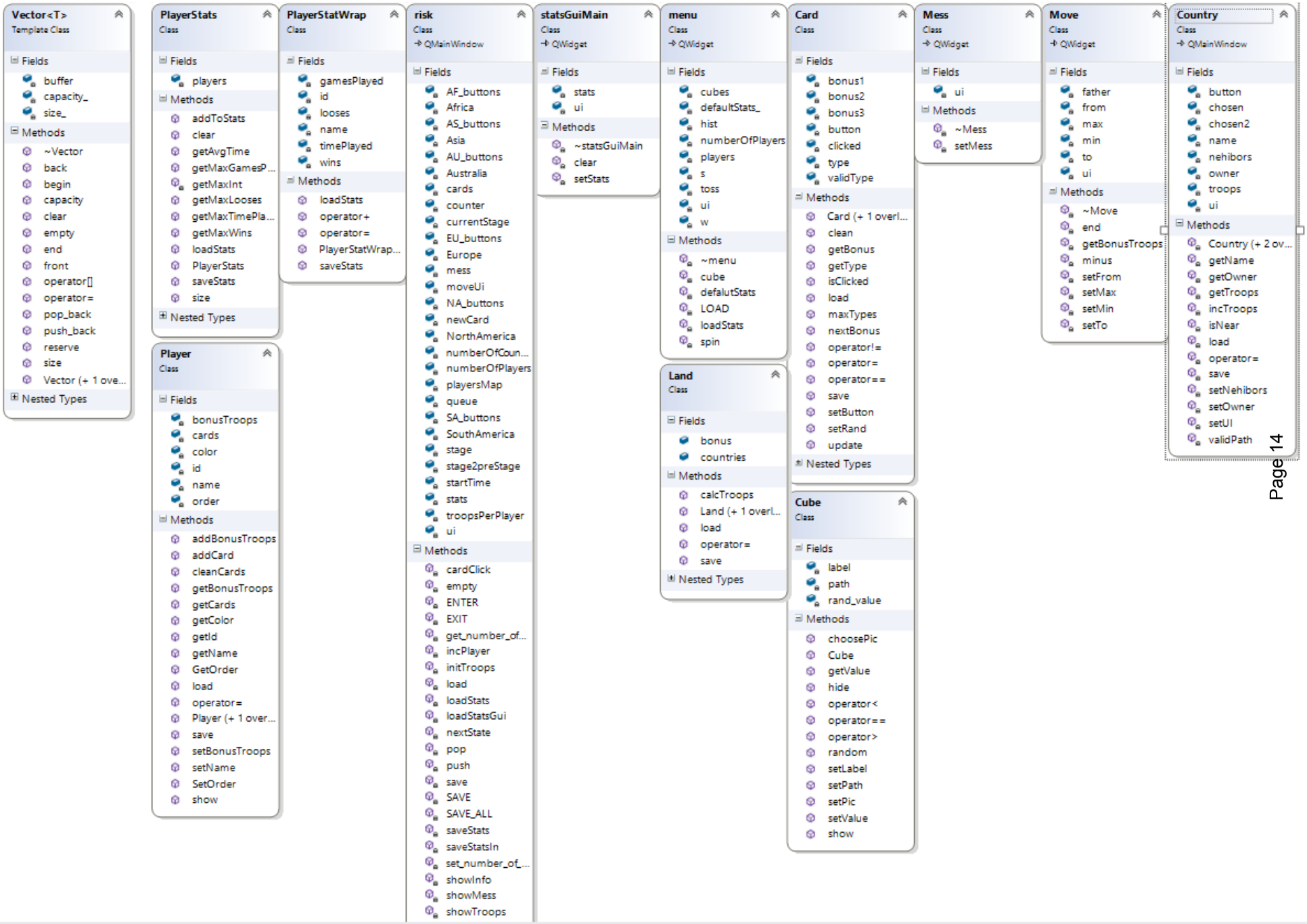
Ctor	Vector();
ייצוג של iterator .	typedef T * iterator;
Copy ctor	Vector(const Vector<T>& v);
הגודל המוקצה של הווקטור –כל פעם שמתמלא מכפיל את עצמו פי 2 בגודל .	unsigned int capacity() const;
כמו מאוכלס בתוך הווקטור	unsigned int size() const;
דגל האומר האם הווקטור ריק	bool empty() const;
מחזיר את האיבר הראשון בווקטור	T& front();
מחזיר את האיבר האחרון בוווקטור	T& back();
מחזיר איטרטורים לתחילה ולסוף של הווקטור	iterator begin(); iterator end();
מכניס איבר לווקטור	void push_back(const T& value);
מוציא איבר מהווקטור	void pop_back();
מקצה מקום בתוך הווקטור מראש	void reserve(unsigned int capacity);
אופרטור []	T & operator[](unsigned int index);
מנקה את הווקטור	void clear();

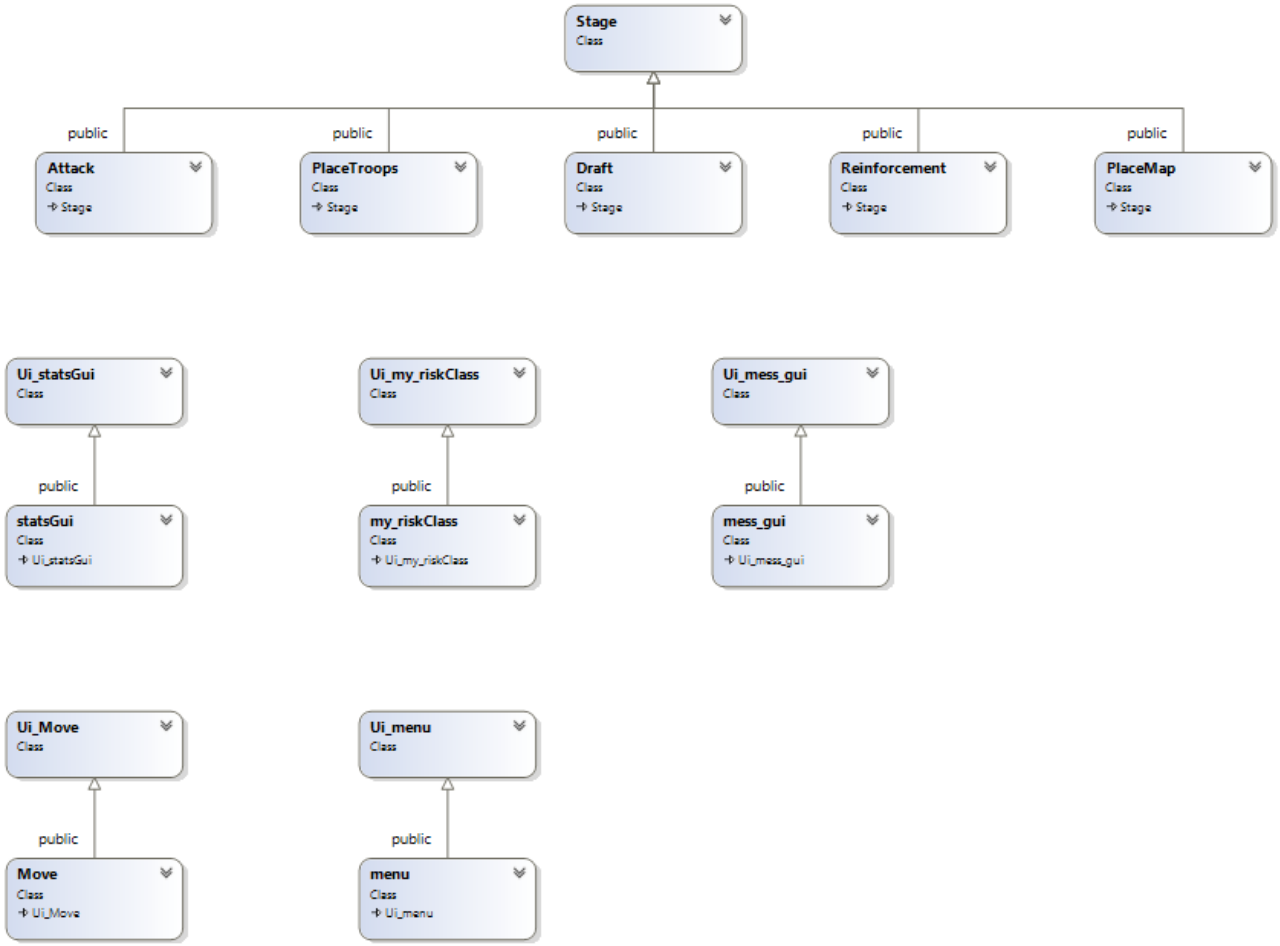
אופרטור =	`ector<T> & operator = (const Vector<T> &);`
	members
מספר האיברים המוקצים בעלי מידע בתוך הווקטור	`unsigned int size_;`
הגודל שאליו מוקצה הווקטור	`unsigned int capacity_;`
המקום בו מאחסן את המידע.	`T* buffer;`

יחסי הגומלין בין המחלקות :



Uml visual studio:





מבני הנתונים בהם השתמשנו:

– Map.1

1. std::map<std::string, std::string>- משמש במשתנה stage2preStage על מנת לקשר בין ה stage הנוכחי ל stage הקודם לצורך טעינת משתנים ותצוגה על ה gui .

2. std::map<std::string, Player>

playersMap –מקשר בין שם השחקן לבין אובייקט של שחקן בעת תהליך הseralization בטעינה מהקובץ.

3 `m3 std::map<std::string, Country*>` -משמש על מנת לקשר בין שם של ארץ לבין הכפתור המקושר אליה- כך ניתן בקלות רבה יותר לקשר כל מדינה אל השכנות שלה- ע"י התאמת אינדקסים של שמות באופן אלגנטי שיקשרו כל מדינה ללחצנים המתאימים של השכנות שלה .

באופן כללי בחרנו להשתמש במבנה נתונים זה כי זמן החיפוש בו הוא מהיר , הוא נוח בקישור של מידע לפריטים במקום להתעסק עם אינדקסים חסרי משמעות מתעסקים עם שמות אינטואיטיביים – דבר אשר הופך את העבודה למאוד אלגנטית וברורה.

queue.2–

`std::queue<Player> queue;`- השתמשנו במבנה נתונים זה על מנת לנהל את התורות של השחקנים במשחק מבנה זה נוח במיוחד לשימוש על מנת לנהל תור של `fifo` כיוון שניתן בקלות רבה להעביר מידע מתחילת התור לסוף התור כפי נדרש לעשות בניהול תור שחקנים.

vector.3 – השתמשנו במבנה נתונים זה על מנת להחזיק את קלפי המשחק שמספר משתנה דימנית וכמו כן להחזיק את השכנים של כל מדינה שגם מספרם הוא משתנה . השימוש בווקטור היה כיוון שהוא מבנה דינמי שמאפשר להרחיבו ולגשת לנתונים בו בקלות רבה. מבנה זה גם מומש לצרוך טעינת הסטטיסטיקות של השחקנים כיוון שאיננו יודעים כמה שחקנים הולכים לטעון.

Seralizition:

בחלק זה יצרנו קובץ שאליו אנחנו שומרים את נתוני המשחק . בחרנו לתת לקובץ את הסיומדת dudu –כינוי ממשחק מחשב, כמובן שהיינו יכולים לתת כל שם אחר.

```
draft
11
-----
2
player1
1234
4
yellow
14
0
-------
player2
5678
1
blue
13
0
-------
Venezuela
1
player2
-----
Brazil
1
player1
-----
Peru
1
player1
-----
```

-השלב במשחק

- counter שסופר עוד כמה חיילים נשאר למקם משמש בשתי השלבים הראשונים של המשחק

-מספר השחקנים

-שם השחקן -,ת.ז, סדר בתור השקחנים ,צבע

מספר חיילי בונוס, מספר קלפים, במידה ויש קלפים – ישמר סוגי הקלפים .

`Joker, Solider, Cannon, Horse, Default`

-הארץ , מספר החיילים הממקומים אליה

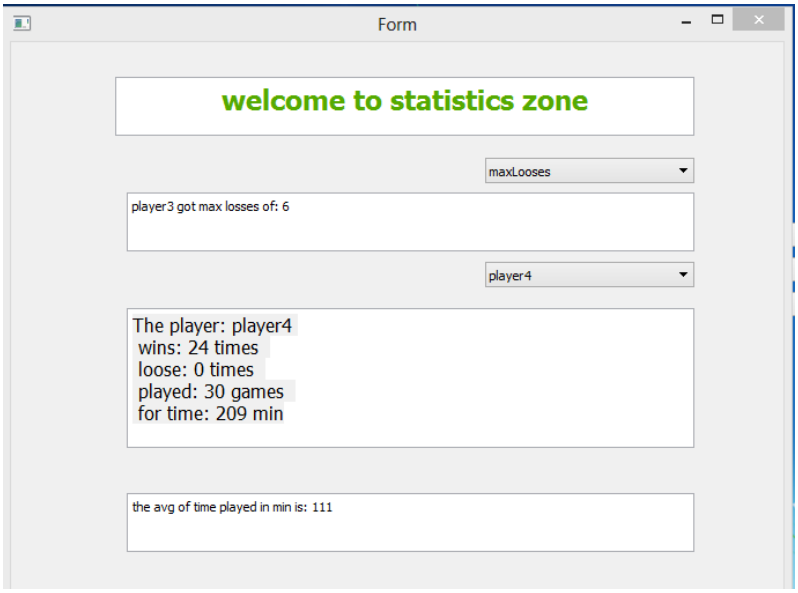
הבעלים שלה

סטטיסטקות :

בתוך המשחק אנחנו שומרים סטטיסטקות כלליות על השחקנים –השחקן עם הכי הרבה נצחונות,השחקן עם הכי הרבה הפסדים השחקן ששיחק את מספר המשחקים הרב ביותר והשחקן ששיחק הכי הרבה זמן כמו כן הזמן הממוצע של המשחק של השחקנים . בתחילת כל משחק ניתן לבחור אם לצפות בסטטיסטיקה הdefault זאת שמתעדכנת במהלך כל משחק או בסטטיסטיקה שנשמרה שלא קשור לסטטיסטיקה הdefault .במהלך המשחק ישנה אפשרות לשמור לקובץ סטטיסטיקה נפרדת שהיא לא ה default – השמירה מתבצעת ע"י לחיצה על לחצן save או ע"י לחיצה על לחצן exit ובמהלך המשחק כל פעם ששחקן מפסיד או מנצח נשמרים נתוניו.
כמו כן בנוסף לנתונים כללים ניתן לראות גם נתונים על כל שחקן ששיחק עד אותו הרגע במשחק.

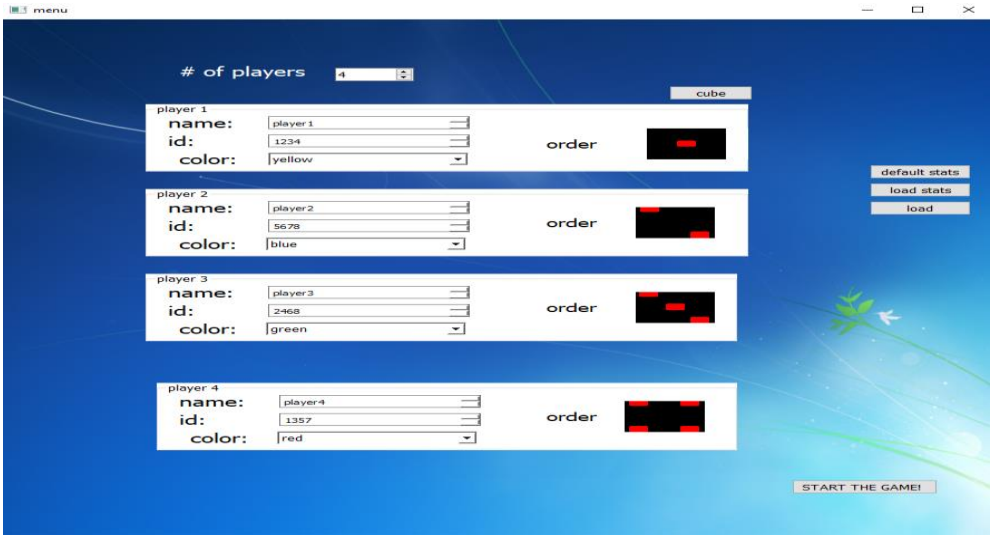
הקובץ ששומר את נתוני המשחק:

שם השחקן	id	נצחונות	הפסדים	זמן משחק(sec)	מספר משחקים
player4	1234	24	0	12592	30
player3	4567	12	6	13096	31
player2	2468	0	0	11944	6
player1	1357	0	0	7239	7
player12	12345	0	0	1488	8
gal	66	0	0	268	0
shoki shalo	1234	0	0	1	1

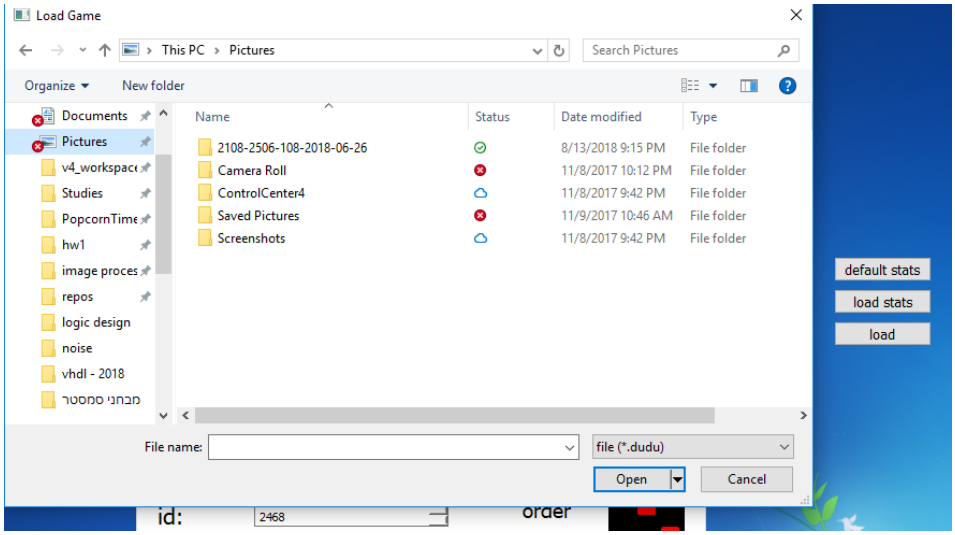


ממשק משתמש :

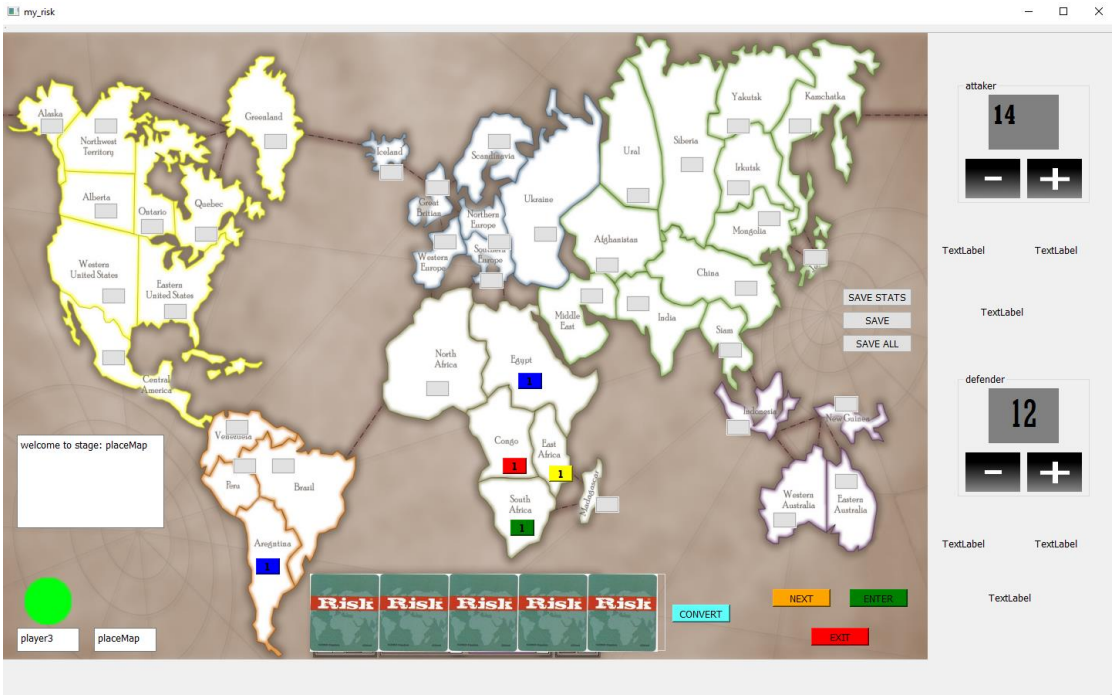
תפריט



בחירת קובץ

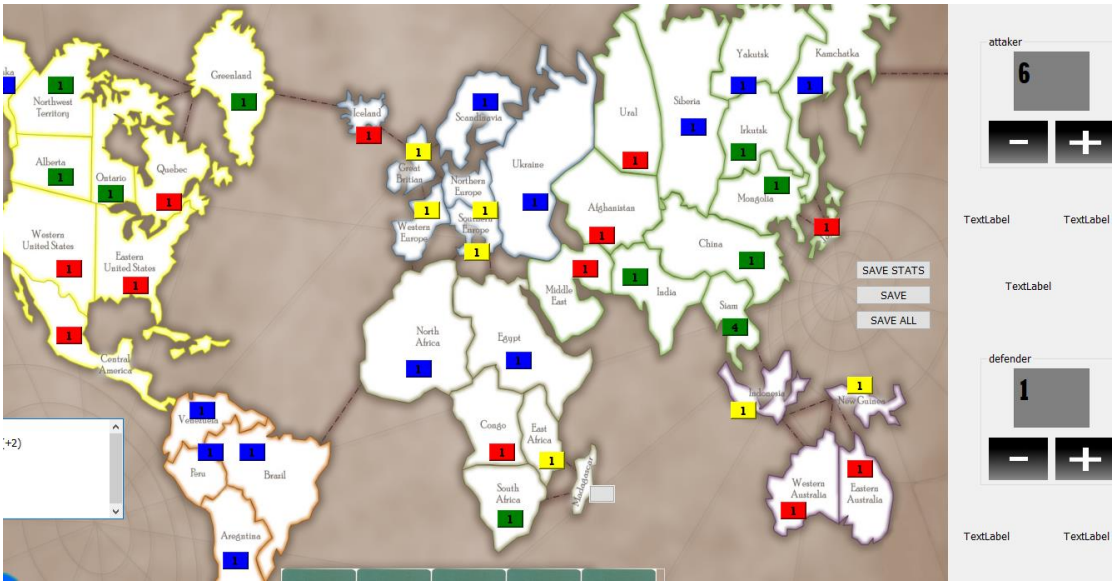


placeMap stage:

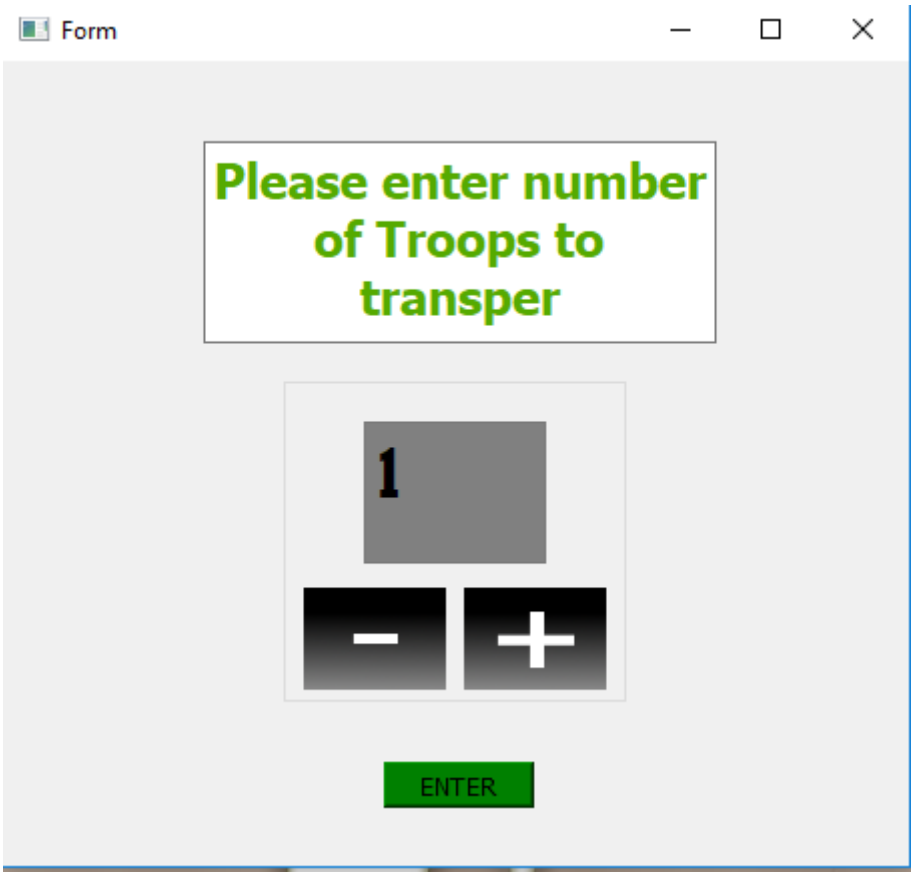


placeTroops stage:

Draft stage:



Move:



delPlayer:

playerWon:

a. מקורות:

- מצגות הקורס – תכנות מכוון אובייקטים 2019 1 – 31695
- Writing of a file: http://www.cplusplus.com/doc/tutorial/files/
- Using static members:
https://www.tutorialspoint.com/cplusplus/cpp_static_members.htm
- Write to CSV file: https://stackoverflow.com/questions/25201131/writing-csv-files-from-c
- STL: https://www.geeksforgeeks.org/the-c-standard-template-library-stl/
- cpp ducmantation - https://en.cppreference.com/w/cpp/container/vector
- qt ducmantation -https://doc.qt.io/-בנוסף לדוקומנטציה נעזרנו הרבה בסרטונים בyoutube המדגימים פעולות שונות בסביבה ובפורם של qt.

main.cpp


```
#include "risk.h"
#include "menu.h"
#include <QtWidgets/QApplication>

int main(int argc, char *argv[])
{
    QApplication a(argc, argv);
    menu m;
    m.show();

    return a.exec();
}

```

menu.h


```
#pragma once
#include <QWidget>
#include "statsGuiMain.h"
#include "risk.h"
#include "Cube.h"
#include <iostream>
class menu : public QWidget {
    Q_OBJECT

public:
    menu(QWidget * parent = Q_NULLPTR);
    ~menu();
private:
    Ui::menu ui;
    risk w;
    statsGuiMain s;
    Cube cubes[4];
    Player players[4];
    Player* hist[6][4];
    int numberOfPlayers;
    bool toss;
    bool defaultStats_;
private slots:
    void end();
    void spin(int);
    void cube();
    void LOAD();
    void loadStats();
    void defalutStats();

};

```

```cpp
#include "menu.h"
#include <time.h>
#include "Cube.h"
#include "windows.h."
#include "qmediaplayer.h"
#include "qmediaplaylist.h"
menu::menu(QWidget * parent) : QWidget(parent), w(4) {
    srand(time(NULL));
    ui.setupUi(this);
    QMediaPlaylist *playlist = new  QMediaPlaylist();
    playlist->addMedia(QUrl("qrc:/new/Resources/Risk.mp3"));
    playlist->setPlaybackMode(QMediaPlaylist::Loop);
    QMediaPlayer* music = new QMediaPlayer();
    music->setPlaylist(playlist);
    music->play();
    this->defaultStats_ = false;
    QPixmap image("Resources/risk_menu.jpg");
    QLabel imageLabel;
    imageLabel.setPixmap(image);
    imageLabel.show();
    this->toss = false;
    this->numberOfPlayers = 4;
    QObject::connect(this->ui.FINISH, &QPushButton::clicked, [=] {end(); });
    QObject::connect(this->ui.cube, &QPushButton::clicked, [=] {cube(); });
    QObject::connect(this->ui.LOAD, &QPushButton::clicked, [=] {LOAD(); });
    QObject::connect(this->ui.loadStats, &QPushButton::clicked, [=] {loadStats(); });
    cubes[0].setLabel(this->ui.cubeP1);
    cubes[1].setLabel(this->ui.cubeP2);
    cubes[2].setLabel(this->ui.cubeP3);
    cubes[3].setLabel(this->ui.cubeP4);
    for (int i = 0; i < 4; i++)
        cubes[i].hide();
    Cube::setPath("Resources/cubes/");
//  this->ui.cubeP4->setScaledContents(true);
    //QObject::connect(this->ui.spinBox, SIGNAL(QSpinBox::valueChanged()), this,SLOT(spin())); -> set in the gui.
    this->ui.colorP1->setCurrentIndex(3);
    this->ui.colorP2->setCurrentIndex(2);
    this->ui.colorP3->setCurrentIndex(1);
    this->ui.colorP4->setCurrentIndex(0);
    this->s.setStats(this->w.loadStats());

    //this->defaultStats_ = true;
    //this->loadStats();
    //this->s.setStats(this->w.loadStats());
    //this->s == nullptr;
}

menu::~menu() {

}
void menu::defalutStats()
{
    this->s.setFocus();
    this->s.show();
```

```cpp

    //this->defaultStats_ = true;
    //this->loadStats();
}
void menu::spin(int numberOfPlayers)
{
    this->numberOfPlayers = numberOfPlayers;

    //HIDE
    if (numberOfPlayers < 4) {
        this->ui.groupBox4->hide();
    }
    if (numberOfPlayers < 3) {
        this->ui.groupBox3->hide();
    }

    //SHOW
    if (numberOfPlayers > 3) {
        this->ui.groupBox4->show();
    }
    if (numberOfPlayers > 2) {
        this->ui.groupBox3->show();
    }

}

void menu::cube()
{
    this->toss = true;
    std::string num2str[] = { "first", "second", "third", "fourth" };
    for (int i = 0; i < 6; i++)
    {
        for (int j = 0; j < 4; j++)
            hist[i][j] = nullptr;
    }
    int iD = std::stoi(this->ui.idP1->toPlainText().toStdString());
    players[0] = Player(this->ui.nameP1->toPlainText().toStdString(), this->ui.colorP1->currentText().toStdString(), 0,iD);
    players[1] = Player(this->ui.nameP2->toPlainText().toStdString(), this->ui.colorP2->currentText().toStdString(), 0,std::stoi(this->ui.idP2->toPlainText().toStdString()));
    players[2] = Player(this->ui.nameP3->toPlainText().toStdString(), this->ui.colorP3->currentText().toStdString(), 0, std::stoi(this->ui.idP3->toPlainText().toStdString()));
    players[3] = Player(this->ui.nameP4->toPlainText().toStdString(), this->ui.colorP4->currentText().toStdString(), 0, std::stoi(this->ui.idP4->toPlainText().toStdString()));
    while (!this->w.empty())
        this->w.pop();

    for (int i = 0; i < this->numberOfPlayers; i++) {
        this->cubes[i].random();
    }
    for (int i = 0; i < this->numberOfPlayers; i++)
    {
        int j = 0;
```

```
        while (hist[this->cubes[i].getValue() - 1][j]) j++;
        hist[this->cubes[i].getValue() - 1][j] = &this->players[i];
    }
    int id = 1;
    for (int i = 5; i >= 0; i--)
    {
        int len = 0;
        while (hist[i][len]) len++;
        if (len == 0) continue;
        while (len)
        {
            hist[i][len-1]->SetOrder(id);
            this->w.push(*hist[i][len-1]);
            id++;
            len--;
        }
    }

    this->ui.pic1_ord->setText(num2str[this->players[0].GetOrder()-1].c_str
      ());
    this->ui.pic2_ord->setText(num2str[this->players[1].GetOrder() - 1].c_str
      ());
    if (this->numberOfPlayers > 2) {
        this->ui.pic3_ord->setText(num2str[this->players[2].GetOrder() -
          1].c_str());
    }
    if (this->numberOfPlayers > 3) {
        this->ui.pic4_ord->setText(num2str[this->players[3].GetOrder() -
          1].c_str());
    }
}

void menu::LOAD()
{
    if (!this->w.LOAD())
        return;
    this->close();
    this->w.setFocus();
    this->w.show();
}

void menu::loadStats()
{
    //if(this->s != nullptr)
    //  this->s.clear();
    if (this->defaultStats_ == true) {
        this->s.setStats(this->w.loadStats());
        this->s.setFocus();
        this->s.show();
        return;
    }
    QString fileName = QFileDialog::getOpenFileName(this,
        tr("Load Game"),
        "",
        tr("file (*.csv);;All Files (*)"));
    if (fileName == "")
```

```
        return;
    this->s.setStats(this->w.loadStats(fileName.toStdString()));
    this->s.setFocus();
    this->s.show();
}


void menu::end() {
    if (this->toss == false) {
        this->ui.info->setText("I see you got NA education .. \nplease toss the cubes!");
        return;
    }
    this->w.set_number_of_player(this->numberOfPlayers);
    this->close();
    this->w.showInfo();
    this->w.initTroops();
    this->w.showTroops();
    this->w.set_startTime(std::time(0));
    this->w.updateStage();
    this->w.setFocus();
    this->w.show();
}
```

risk.h


```
1  #pragma once
2  #include "Player.h"
3  #include "Country.h"
4  #define num_of_conutries 42
5  #define TOTAL_TROOPS 100
6  #define TROOP4COUNTRY 3
7  #include <QtWidgets/QMainWindow>
8  #include <QMediaPlayer>
9  #include "ui_my_risk.h"
10 #include "Qdebug.h"
11 #include <Qsignalmapper.h>
12 #include <utility>
13 #include <queue>
14 #include <fstream>
15 #include <qfiledialog.h>
16 #include <functional>
17 #include <iostream>
18 #include "Cube.h"
19 #include "Move.h"
20 #include "Stage.h"
21 #include "Attack.h"
22 #include "Draft.h"
23 #include "Reinforcement.h"
24 #include "PlaceMap.h"
25 #include "PlaceTroops.h"
26 #include "PlayerStats.h"
27 #include <chrono>
28 #include <ctime>
29 #include "Mess.h"
30 #define _maxCardsPerPlayer_ 5
31 class risk :public QMainWindow {
32     Q_OBJECT
33 public:
34     risk(int numberOfPlayers, QWidget *parent = 0);
35     char get_number_of_player();
36     void set_number_of_player(char num);
37     void incPlayer();
38     void initTroops();
39     void showTroops();
40
41     bool empty() const;
42     void pop();
43     void push(const Player&);
44
45     void save(std::ofstream& file);
46     void load(std::ifstream& file);
47     PlayerStats* loadStats(std::string fileName = "Resources/stats.csv");
48     void loadStatsGui();
49     void showInfo();
50     void updateStage();
51     time_t get_startTime() const;
52     void set_startTime(const time_t);
53 private:
54     std::queue<Player> queue;
55     Land SouthAmerica, NorthAmerica, Europe, Asia, Australia, Africa;
56     std::vector<QPushButton*> SA_buttons;
```

```
    std::vector<QPushButton*> NA_buttons;
    std::vector<QPushButton*> EU_buttons;
    std::vector<QPushButton*> AS_buttons;
    std::vector<QPushButton*> AU_buttons;
    std::vector<QPushButton*> AF_buttons;

    char numberOfPlayers;
    int numberOfCountries;  //set in ctor and does not change
    int troopsPerPlayer;
    int counter;    //counter in placeMap and placeTroops()
    bool newCard;
    std::vector<Card> cards;
    void cardClick(int sel = 0);

    std::string stage;
    Stage* currentStage;
    std::map<std::string, std::string> stage2preStage;
    PlayerStats stats;
    time_t startTime;
    void showMess(std::string);
    Mess mess;
    std::map<std::string, Player> playersMap;
    Move moveUi;
    void saveStatsIn(std::string = "Resources/stats.csv",bool all = 0);
public slots:
    bool LOAD();
    void SAVE();
    void ENTER();
    void SAVE_ALL();
    void saveStats();
    void EXIT();
    void nextState();

private:
    Ui::my_riskClass ui;
};
```

```cpp
#include "risk.h"

risk::risk(int numberOfPlayers, QWidget *parent) :troopsPerPlayer (TOTAL_TROOPS / numberOfPlayers), QMainWindow(parent), cards { Card::cardType::Default, Card::cardType::Default, Card::cardType::Default, Card::cardType::Default,Card::cardType::Default }
{
    ui.setupUi(this);
    //SouthAmerica, NorthAmerica, Europe, Asia, Australia, Africa;
    Country::setUI(&this->ui);
    newCard = false;
    this->numberOfPlayers = numberOfPlayers;
    this->counter = 0;
    this->stage = "placeMap";
    QObject::connect(this->ui.ENTER, &QPushButton::clicked, [=] {ENTER(); });
    QObject::connect(this->ui.STATS, &QPushButton::clicked, [=] {saveStats();});

    //this->currentStage->setNextEn(true);
    stage2preStage["placeMap"] = "initial";
    stage2preStage["placeTroops"] = "placeMap";
    stage2preStage["draft"] = "placeTroops";
    stage2preStage["attack"] = "draft";
    stage2preStage["reinforcement"] = "attack";
    stage2preStage["default"] = "reinforcement";


    this->cards[0].setButton(this->ui.card1);
    this->cards[1].setButton(this->ui.card2);
    this->cards[2].setButton(this->ui.card3);
    this->cards[3].setButton(this->ui.card4);
    this->cards[4].setButton(this->ui.card5);
    this->ui.cubeP_1->hide();
    this->ui.cubeP_2->hide();
    this->ui.cubeP_3->hide();
    this->ui.cubeP2_1->hide();
    this->ui.cubeP2_2->hide();
    this->ui.cubeP2_3->hide();
    this->ui.TROOPS_2->setVisible(false);

    ////////////////////////////////    SA    ////////////////////////////////
    std::vector< std::string> SA{ "Venezuela","Brazil","Peru","Argentina" };
    this->SouthAmerica = Land(SA, { 2,"WellDone you conquered SouthAmerica (+2)" });

    this->SA_buttons.reserve(SA.size());
    this->SA_buttons = {
        this->ui.south_america_1 ,this->ui.south_america_2 ,this->ui.south_america_3,
        this->ui.south_america_4
    };

    for (int i = 0; i < this->SA_buttons.size(); ++i) {
        SouthAmerica.countries[i].setButton(this->SA_buttons[i]);
        QObject::connect(this->SA_buttons[i], &QPushButton::clicked, [=] {SouthAmerica.countries[i].init(); });
```

```
    }

    ////////////////////////////////    NA    ////////////////////////////////
    std::vector< std::string> NA{ "Alaska","Northwest Territory","Greenland","Alberta","ontario","Quebec","Western United States","Eastern United States","Central America" };
    this->NorthAmerica = Land(NA, { 5 ,"WellDone you conquered NorthAmerica (+5) - about 7.5% of the world's population\n"});

    this->NA_buttons.reserve(NA.size());
    this->NA_buttons = {
        this->ui.north_america_1 ,this->ui.north_america_2 ,this->ui.north_america_3,
        this->ui.north_america_4 ,this->ui.north_america_5 ,this->ui.north_america_6,
        this->ui.north_america_7 ,this->ui.north_america_8 ,this->ui.north_america_9
    };

    for (int i = 0; i < this->NA_buttons.size(); ++i) {
        NorthAmerica.countries[i].setButton(this->NA_buttons[i]);
        QObject::connect(this->NA_buttons[i], &QPushButton::clicked, [=] {NorthAmerica.countries[i].init(); });
    }

    ////////////////////////////////    AF    ////////////////////////////////
    std::vector< std::string> AF{ "North Africa" ,"Egypt" ,"Cango" ,"East Africa","South Africa","Madagascar" };
    this->Africa = Land(AF,{3, "WellDone you conquered Africa(+3)"});

    this->AF_buttons.reserve(AF.size());
    this->AF_buttons = {
        this->ui.africa ,this->ui.africa_2 ,this->ui.africa_3,
        this->ui.africa_4 ,this->ui.africa_5 ,this->ui.africa_6
    };

    for (int i = 0; i < this->AF_buttons.size(); ++i) {
        Africa.countries[i].setButton(this->AF_buttons[i]);
        QObject::connect(this->AF_buttons[i], &QPushButton::clicked, [=] {Africa.countries[i].init(); });
    }

    ////////////////////////////////    EU    ////////////////////////////////
    std::vector< std::string> EU{ "Iceland","Great Berlin","Scandinavia","Western Europe" ,"Soutern Europe" ,"Northern Europe" ,"Ukraine" };
    this->Europe = Land(EU, { 5,"WellDone you conquered Europe (+5), did you know 28 of the European states belong to the European Union?" });

    this->EU_buttons.reserve(EU.size());
    this->EU_buttons = {
        this->ui.europe_1 ,this->ui.europe_2 ,this->ui.europe_3,
        this->ui.europe_4 ,this->ui.europe_5 ,this->ui.europe_6,this->ui.europe_7
    };

```

```cpp
    for (int i = 0; i < this->EU_buttons.size(); ++i) {
        Europe.countries[i].setButton(this->EU_buttons[i]);
        QObject::connect(this->EU_buttons[i], &QPushButton::clicked, [=] {Europe.countries[i].init(); });
    }

    ////////////////////////////////    AS    ////////////////////////////////
    std::vector< std::string> AS { "Kamchatka","Yakutsk","Siberia","Irkutsk" ,"Mongolia" ,"China" ,"India","Siam","Ural","Afghanistan","Middle East","Japan" };
    this->Asia = Land(AS,{7, "WellDone you conquered Asia (+5)(+7)"});

    this->AS_buttons.reserve(AS.size());
    this->AS_buttons = {
        this->ui.asia ,this->ui.asia_2,this->ui.asia_3,this->ui.asia_4,this->ui.asia_5,
        this->ui.asia_6 ,this->ui.asia_7 ,this->ui.asia_8,this->ui.asia_9,this->ui.asia_10,this->ui.asia_11, this->ui.asia_12
    };

    for (int i = 0; i < this->AS_buttons.size(); ++i) {
        Asia.countries[i].setButton(this->AS_buttons[i]);
        QObject::connect(this->AS_buttons[i], &QPushButton::clicked, [=] {Asia.countries[i].init(); });
    }
    ////////////////////////////////    AU    ////////////////////////////////
    std::vector< std::string> AU{ "Western Australia" ,"Eastern Australia" ,"New Gulinea" ,"Indonesia" };
    this->Australia = Land(AU, { 2,"WellDone you conquered Australia (+2)" });

    this->AU_buttons.reserve(AU.size());
    this->AU_buttons = {
        this->ui.australia ,this->ui.australia_2,
        this->ui.australia_3,this->ui.australia_4
    };
    for (int i = 0; i < this->AU_buttons.size(); ++i) {
        Australia.countries[i].setButton(this->AU_buttons[i]);
        QObject::connect(this->AU_buttons[i], &QPushButton::clicked, [=] {Australia.countries[i].init(); });
    }
    this->numberOfCountries = AU.size() + AS.size() + NA.size() + SA.size() + AF.size() + EU.size();
    //set neihbors
    /*
    */
    // in order to orginaze the connection easily we map the country names temprarly to pointer to their objects.

    std::map<std::string, Country*> m;
    m["Alaska"] = &(this->NorthAmerica.countries[0]);
    m["Alberta"] = &(this->NorthAmerica.countries[3]);
    m["Central America"] = &(this->NorthAmerica.countries[8]);
    m["Eastern United States"] = &(this->NorthAmerica.countries[7]);
    m["Greenland"] = &(this->NorthAmerica.countries[2]);
    m["Northwest Territory"] = &(this->NorthAmerica.countries[1]);
    m["Ontario"] = &(this->NorthAmerica.countries[4]);
```

```
    m["Quebec"] = &(this->NorthAmerica.countries[5]);
    m["Western United States"] = &(this->NorthAmerica.countries[6]);

    /////////////////
    m["Argentina"] = &(this->SouthAmerica.countries[3]);
    m["Brazil"] = &(this->SouthAmerica.countries[1]);
    m["Peru"] = &(this->SouthAmerica.countries[2]);
    m["Venezuela"] = &(this->SouthAmerica.countries[0]);
    ////////////////////
    m["Great Britain"] = &(this->Europe.countries[1]);
    m["Iceland"] = &(this->Europe.countries[0]);
    m["Northern Europe"] = &(this->Europe.countries[5]);
    m["Scandinavia"] = &(this->Europe.countries[2]);
    m["Southern Europe"] = &(this->Europe.countries[4]);
    m["Ukraine"] = &(this->Europe.countries[6]);
    m["Western Europe"] = &(this->Europe.countries[3]);
    ///////////////
    m["Cango"] = &(this->Africa.countries[2]);
    m["East Africa"] = &(this->Africa.countries[3]);
    m["Egypt"] = &(this->Africa.countries[1]);
    m["Madagascar"] = &(this->Africa.countries[5]);
    m["North Africa"] = &(this->Africa.countries[0]);
    m["South Africa"] = &(this->Africa.countries[4]);
    //std::vector< std::string> AS{ "Kamchatka0", "Yakutsk1", "Siberia2", "Irkutsk3", "Mongolia4", "China5", "India6", "Siam7", "Ural8", "Afghanistan9", "Middle East10",japen[11] }; Africa
    m["Afghanistan"] = &(this->Asia.countries[9]);
    m["China"] = &(this->Asia.countries[5]);
    m["India"] = &(this->Asia.countries[6]);
    m["Irkutsk"] = &(this->Asia.countries[3]);
    m["Japan"] = &(this->Asia.countries[11]);
    m["Kamchatka"] = &(this->Asia.countries[0]);
    m["Middle East"] = &(this->Asia.countries[10]);
    m["Mongolia"] = &(this->Asia.countries[4]);
    m["Siam"] = &(this->Asia.countries[7]);
    m["Siberia"] = &(this->Asia.countries[2]);
    m["Ural"] = &(this->Asia.countries[8]);
    m["Yakutsk"] = &(this->Asia.countries[1]);

    //std::vector< std::string> AU{ "Western Australia", "Eastern Australia", "New Gulinea", "Indonesia" }; Australia
    m["Eastern Australia"] = &(this->Australia.countries[0]);
    m["Indonesia"] = &(this->Australia.countries[3]);
    m["New Guinea"] = &(this->Australia.countries[2]);
    m["Western Australia"] = &(this->Australia.countries[1]);


    //SA
    m["Argentina"]->setNehibors({ m["Brazil"], m["Peru"] });
    m["Brazil"]->setNehibors({ m["Peru"], m["Venezuela"], m["Argentina"], m["North Africa"] });
    m["Peru"]->setNehibors({ m["Venezuela"], m["Brazil"] ,m["Argentina"] });
    m["Venezuela"]->setNehibors({ m["Central America"], m["Peru"], m["Brazil"] });
    //eu
    m["Great Britain"]->setNehibors({ m["Iceland"], m["Scandinavia"], m
```

```
["Western Europe"], m["Northern Europe"] });
    m["Iceland"]->setNehibors({ m["Greenland"], m["Great Britain"] });
    m["Northern Europe"]->setNehibors({ m["Ukraine"], m["Soutern Europe"],m["Western Europe"],m["Great Britain"] });
    m["Scandinavia"]->setNehibors({ m["Ukraine"], m["Great Britain"] });
    m["Southern Europe"]->setNehibors({ m["North Africa"], m["Egypt"] ,m["Ukraine"] ,m["Western Europe"],m["Northern Europe"] });
    m["Ukraine"]->setNehibors({ m["Scandinavia"], m["Northern Europe"] ,m["Southern Europe"] ,m["Middle East"], m["Ural"],m["Afghanistan"] });
    m["Western Europe"]->setNehibors({ m["North Africa"], m["Southern Europe"] , m["Northern Europe"] , m["Great Britain"] });
    //af
    m["North Africa"]->setNehibors({ m["Brazil"], m["Cango"],m["Egypt"], m["Southern Europe"],  m["Western Europe"] });
    m["Egypt"]->setNehibors({ m["Middle East"], m["Southern Europe"], m["North Africa"], m["Cango"],m["East Africa"] });
    m["Cango"]->setNehibors({ m["South Africa"], m["East Africa"] ,m["Egypt"] ,m["North Africa"] });
    m["East Africa"]->setNehibors({ m["Cango"], m["Egypt"], m["South Africa"],m["Madagascar"] });
    m["South Africa"]->setNehibors({ m["Madagascar"], m["East Africa"] ,m["Cango"] });
    m["Madagascar"]->setNehibors({ m["East Africa"], m["South Africa"] });
    //as
    m["Afghanistan"]->setNehibors({ m["Ukraine"], m["Middle East"] , m["India"] , m["China"] , m["Ural"] , m["Siberia"] });
    m["China"]->setNehibors({ m["India"], m["Siam"], m["Afghanistan"] , m["Mongolia"]  , m["Siberia"] , m["Ural"] });
    m["India"]->setNehibors({ m["Middle East"], m["Afghanistan"], m["China"], m["Siam"] });
    m["Irkutsk"]->setNehibors({ m["Mongolia"], m["Siberia"] , m["Yakutsk"] , m["Kamchatka"] });
    m["Japan"]->setNehibors({ m["Mongolia"], m["Kamchatka"] });
    m["Kamchatka"]->setNehibors({ m["Irkutsk"], m["Yakutsk"] , m["Mongolia"], m["Japan"],m["Alaska"] });
    m["Middle East"]->setNehibors({ m["India"], m["Afghanistan"] , m["Ukraine"], m["Egypt"] });
    m["Mongolia"]->setNehibors({ m["Japan"], m["China"] , m["Irkutsk"] , m["Siberia"] });
    m["Siam"]->setNehibors({ m["China"], m["India"] , m["Indonesia"] });
    m["Siberia"]->setNehibors({ m["Yakutsk"], m["Irkutsk"], m["Mongolia"], m["China"] , m["Afghanistan"] , m["Ural"] });
    m["Ural"]->setNehibors({ m["Siberia"], m["Ukraine"] , m["Afghanistan"] });
    m["Yakutsk"]->setNehibors({ m["Kamchatka"], m["Irkutsk"], m["Siberia"] });
    //au
    m["Eastern Australia"]->setNehibors({ m["Western Australia"], m["New Guinea"] });
    m["Indonesia"]->setNehibors({ m["Siam"], m["New Guinea"], m["Western Australia"] });
    m["New Guinea"]->setNehibors({ m["Indonesia"], m["Eastern Australia"], m["Western Australia"] });
    m["Western Australia"]->setNehibors({ m["New Guinea"], m["Indonesia"], m["Eastern Australia"] });

    //na
    m["Alaska"]->setNehibors({ m["Northwest Territory"], m["Kamchatka"] });
```

```
    m["Alberta"]->setNehibors({ m["Northwest Territory"], m["Ontario"], m["Western United States"], m["Alaska"] });
    m["Central America"]->setNehibors({ m["Venezuela"], m["Eastern United States"], m["Western United States"] });
    m["Eastern United States"]->setNehibors({ m["Quebec"], m["Ontario"], m["Western United States"], m["Central America"] });
    m["Greenland"]->setNehibors({ m["Northwest Territory"], m["Ontario"], m["Quebec"], m["Iceland"] });
    m["Northwest Territory"]->setNehibors({ m["Alaska"], m["Alberta"], m["Ontario"] });
    m["Ontario"]->setNehibors({ m["Northwest Territory"], m["Alberta"], m["Western United State"], m["Eastern United States"] , m["Quebec"] });
    m["Quebec"]->setNehibors({ m["Ontario"], m["Greenland"], m["Eastern United States"] });
    m["Western United States"]->setNehibors({ m["Alberta"], m["Eastern United States"], m["Ontario"], m["Central America"] });



    this->queue.push(Player());
    this->currentStage = new PlaceMap(this->ui, this->queue.front(), this->moveUi, this->counter);
    ui.textEdit->setText("welocme to the game");
}


void risk::incPlayer()
{
    this->queue.push(this->queue.front());
    this->queue.pop();
    QString picPath = QString("Resources/players/") + QString(this->queue.front().getColor().c_str()) + ".png";
    QPixmap pic(picPath);//("Resources/cubes/1.png");
    this->ui.colorP->setPixmap(pic);
    this->ui.colorP->show();
}

void risk::cardClick(int sel)
{
    this->cards[sel].click();
}

void risk::showMess(std::string txt)
{
    this->mess.setMess(txt);
    this->mess.setFocus();
    this->mess.show();
}

void risk::showTroops()
{
    int bonus = this->queue.front().getBonusTroops();
    QString text = std::to_string(bonus).c_str();
    this->ui.troopsP->setText(text);
}

```

```cpp
bool risk::empty() const
{
    return this->queue.empty();
}

void risk::pop()
{
    this->queue.pop();
}

void risk::push(const Player & player)
{
    this->queue.push(player);
}

void risk::initTroops()
{
    //y=-5x+50;     -> this is the real but it is TOO MUCH!
    int troops = -5 * this->numberOfPlayers + 35;
    for (int i = 0; i < this->numberOfPlayers; i++) {
        this->queue.front().setBonusTroops(troops);
        incPlayer();
    }
}

void risk::nextState()
{
    if (this->currentStage->getNexEn() == false)
        return;
    this->currentStage->afterAction();
    if (this->currentStage->getNexEn() == false)//case  draft that you have cards to convert
        return;
    delete this->currentStage;
    if (this->stage == "initial") {
        this->stage = "placeMap";
        this->currentStage = new PlaceMap(this->ui, this->queue.front(), this->moveUi, this->counter);
    }
    else if (this->stage == "placeMap") {
        this->stage = "placeTroops";
        this->currentStage = new PlaceTroops(this->ui, this->queue.front(), this->moveUi, this->counter);
    }
    else if (this->stage == "placeTroops") {
        this->stage = "draft";
        this->currentStage = new Draft(this->ui, this->queue.front(), this->moveUi, this->cards,_maxCardsPerPlayer_);
        std::vector<Land*> lands = { &this->SouthAmerica,
                                     &this->NorthAmerica,
                                     &this->Asia,
                                     &this->Australia,
                                     &this->Africa,
                                     &this->Europe,
        };
        dynamic_cast<Draft*>(this->currentStage)->setLands(lands);
```

```cpp

        this->ui.PLUS->disconnect();
        this->ui.MINUS->disconnect();
        QObject::connect(this->ui.PLUS, &QPushButton::clicked, [=] {currentStage->plus(); });
        QObject::connect(this->ui.MINUS, &QPushButton::clicked, [=] {currentStage->minus(); });

        this->ui.card1->disconnect();
        this->ui.card2->disconnect();
        this->ui.card3->disconnect();
        this->ui.card4->disconnect();
        this->ui.card5->disconnect();

        QObject::connect(this->ui.card1, &QPushButton::clicked, [=] {cards[0].click(); });
        QObject::connect(this->ui.card2, &QPushButton::clicked, [=] {cards[1].click(); });
        QObject::connect(this->ui.card3, &QPushButton::clicked, [=] {cards[2].click(); });
        QObject::connect(this->ui.card4, &QPushButton::clicked, [=] {cards[3].click(); });
        QObject::connect(this->ui.card5, &QPushButton::clicked, [=] {cards[4].click(); });

        this->ui.CONVERT->disconnect();
        QObject::connect(this->ui.CONVERT, &QPushButton::clicked, [=] {dynamic_cast<Draft*>(currentStage)->convert(); });
    }
    else if (this->stage == "draft") {
        this->stage = "attack";
        this->ui.CONVERT->disconnect();
        int idx = 0;
        while (this->cards[idx++] != Card(Card::Default));
        idx--;
        this->currentStage = new Attack(this->ui, this->queue.front(), this->moveUi, &(this->cards[idx]));
        this->ui.PLUS->disconnect();
        this->ui.PLUS_2->disconnect();
        this->ui.MINUS->disconnect();
        this->ui.MINUS_2->disconnect();

        QObject::connect(this->ui.PLUS, &QPushButton::clicked, [=] {currentStage->plus(); });
        QObject::connect(this->ui.MINUS, &QPushButton::clicked, [=] {currentStage->minus(); });
        QObject::connect(this->ui.PLUS_2, &QPushButton::clicked, [=] {currentStage->plus(1); });
        QObject::connect(this->ui.MINUS_2, &QPushButton::clicked, [=] {currentStage->minus(1); });

    }
    else if (this->stage == "attack") {
        this->currentStage = new Reinforcement(this->ui, this->queue.front(), this->moveUi);
        this->ui.PLUS->disconnect();
```

```
        this->ui.PLUS_2->disconnect();
        this->ui.MINUS->disconnect();
        this->ui.MINUS_2->disconnect();

        QObject::connect(this->ui.PLUS, &QPushButton::clicked, [=] {currentStage->plus(); });
        QObject::connect(this->ui.MINUS, &QPushButton::clicked, [=] {currentStage->minus(); });
        QObject::connect(this->ui.PLUS_2, &QPushButton::clicked, [=] {currentStage->plus(1); });
        QObject::connect(this->ui.MINUS_2, &QPushButton::clicked, [=] {currentStage->minus(1); });

        this->stage = "reinforcement";
    }
    else if (this->stage == "reinforcement") {
        this->stage = "draft";
        this->incPlayer();
        this->currentStage = new Draft(this->ui, this->queue.front(), this->moveUi, this->cards, _maxCardsPerPlayer_);
        std::vector<Land*> lands = { &this->SouthAmerica,
                                     &this->NorthAmerica,
                                     &this->Asia,
                                     &this->Australia,
                                     &this->Africa,
                                     &this->Europe,
        };
        dynamic_cast<Draft*>(this->currentStage)->setLands(lands);

        Land::landInfo res = dynamic_cast<Draft*>(this->currentStage)->calcTroops();
        if (res.bonus == -1) {  //player need to be delete.
            this->numberOfPlayers--;
            if (this->numberOfPlayers == 1) {   //won!
                this->queue.pop();
                this->showMess(this->queue.front().getName() + " won!");
                this->saveStatsIn();
            }
            else {
                this->showMess(this->queue.front().getName() + " kicked out! losser..");
                this->saveStatsIn();
                //this->saveStats();
                this->queue.pop();
                return this->updateStage();
            }
        }
        this->ui.PLUS->disconnect();
        this->ui.PLUS_2->disconnect();
        this->ui.MINUS->disconnect();
        this->ui.MINUS_2->disconnect();

        QObject::connect(this->ui.PLUS, &QPushButton::clicked, [=] {currentStage->plus(); });
        QObject::connect(this->ui.MINUS, &QPushButton::clicked, [=] {currentStage->minus(); });
```

```
        QObject::connect(this->ui.PLUS_2, &QPushButton::clicked, [=] {currentStage->plus(1); });
        QObject::connect(this->ui.MINUS_2, &QPushButton::clicked, [=] {currentStage->minus(1); });


        this->ui.card1->disconnect();
        this->ui.card2->disconnect();
        this->ui.card3->disconnect();
        this->ui.card4->disconnect();
        this->ui.card5->disconnect();

        QObject::connect(this->ui.card1, &QPushButton::clicked, [=] {cards[0].click(); });
        QObject::connect(this->ui.card2, &QPushButton::clicked, [=] {cards[1].click(); });
        QObject::connect(this->ui.card3, &QPushButton::clicked, [=] {cards[2].click(); });
        QObject::connect(this->ui.card4, &QPushButton::clicked, [=] {cards[3].click(); });
        QObject::connect(this->ui.card5, &QPushButton::clicked, [=] {cards[4].click(); });

        this->ui.CONVERT->disconnect();
        QObject::connect(this->ui.CONVERT, &QPushButton::clicked, [=] {dynamic_cast<Draft*>(currentStage)->convert(); });
    }
    this->currentStage->preAction();
    this->showInfo();
    this->ui.playerText->setText(this->queue.front().getName().c_str());
    this->ui.stageText->setText(this->stage.c_str());
}

void risk::save(std::ofstream& file) {
    file << this->stage << "\n";
    file << this->counter << "\n";
    file << "-----\n";
    file << std::to_string(this->numberOfPlayers) << "\n";
    std::queue<Player> queue = this->queue;
    for (int i = 0; i < this->numberOfPlayers; i++) {
        queue.front().save(file);
        queue.push(this->queue.front());
        queue.pop();
    }
    this->SouthAmerica.save(file);
    this->NorthAmerica.save(file);
    this->Europe.save(file);
    this->Asia.save(file);
    this->Australia.save(file);
    this->Africa.save(file);
}

void risk::load(std::ifstream & file)
{
    std::string tmp;
    //load stage
```

```
    file >> this->stage;
    file >> tmp;
    this->counter = std::stoi(tmp);
    file >> tmp;    //dummy

    this->ui.textEdit->setText(QString("welcome to stage:  ") + this->stage.c_str()+ QString("you need to place all your troops in this stage "));
    //load players
    file >> tmp;
    this->numberOfPlayers = std::stoi(tmp);

    while (!this->queue.empty())
        this->queue.pop();

    for (int i = 0; i < this->numberOfPlayers; i++) {
        Player tmp;
        tmp.load(file);
        this->playersMap[tmp.getName()] = tmp;
        this->queue.push(tmp);
    }
    //load Lands
    this->SouthAmerica.load(file, this->playersMap);
    this->NorthAmerica.load(file, this->playersMap);
    this->Europe.load(file, this->playersMap);
    this->Asia.load(file, this->playersMap);
    this->Australia.load(file, this->playersMap);
    this->Africa.load(file, this->playersMap);

    this->updateStage();
}

void risk::saveStats()
{
    QString fileName = QFileDialog::getSaveFileName(this,
        tr("Save Stats"),
        "",
        tr("file (*.csv);;All Files (*)"));

    this->saveStatsIn(fileName.toStdString(),1);
}

void risk::EXIT()
{
    if (this->numberOfPlayers == 1)
    {
        this->saveStatsIn("Resources/stats.csv", 0);
        exit(0);
    }
    this->saveStatsIn("Resources/stats.csv", 1);
    exit(0);
}

PlayerStats* risk::loadStats(std::string fileName)
{
    std::ifstream file;
```

```
    file.open(fileName);
    this->stats.loadStats(file);
    file.close();
    return &this->stats;
}

void risk::loadStatsGui()
{
    QString fileName = QFileDialog::getOpenFileName(this,
    tr("Load Game"),
        "",
        tr("file (*.csv);;All Files (*)"));
    this->loadStats(fileName.toStdString());
}

void risk::showInfo()
{
    QString picPath = QString("Resources/players/") + QString(this->queue.front().getColor().c_str()) + ".png";
    QPixmap pic(picPath);//("Resources/cubes/1.png");
    this->ui.colorP->setPixmap(pic);
    this->ui.colorP->show();
}

void risk::updateStage()
{
    this->stage = this->stage2preStage[this->stage];
    this->currentStage->setNextEn(true);
    this->nextState();
}

time_t risk::get_startTime() const
{
    return this->startTime;
}

void risk::set_startTime(const time_t time )
{
    this->startTime = time;
}

void risk::SAVE_ALL()
{
    this->SAVE();
    this->saveStats();
}
///this function is called in some situation -first it called each time player lose or win.
// second it called when player clicked exit butten.
// it also called whether the path for save it the default and when it isn't mean the player chose other path
// all variable is flag that send in order to know if the function is called with exit
// then no one lose or win or when player is kicked out or win during the game.
//all is 0 by default and its called with 1 only when clicked exit and its
```

```
    only 1 player-means win.
void risk::saveStatsIn(std::string fileName, bool all)
{
    std::ofstream file;
    file.open(fileName);
    std::queue<Player> queue = this->queue;
    time_t end = std::time(0);
    int elapsed_seconds = end - this->startTime;
    this->startTime = end;
    int len = all ? this->numberOfPlayers : 1;
    int win = 0;    //nutral
    if (this->numberOfPlayers == 1)
        win = 1;    //win
    else if (all == false)
        win = -1; //loose
    for (int i = 0; i < len; ++i) {
        PlayerStatWrap tmp(queue.front(), win, elapsed_seconds);//
          player,win,time

        this->stats.addToStats(tmp);
        queue.pop();
    }
    this->stats.saveStats(file);
    file.close();
}

void risk::SAVE()
{

    QString fileName = QFileDialog::getSaveFileName(this,
        tr("Save Game"),
        "",
        tr("file (*.dudu);;All Files (*)"));
    std::ofstream file;
    file.open(fileName.toStdString());
    this->save(file);
    file.close();

}

void risk::ENTER()
{
    if (Country::chosen == nullptr)
    {
        this->ui.textEdit->setText(QString("please start the game .."));
        return;
    }
    this->currentStage->action();
    if (this->currentStage->getEnterEn() == true && (this->stage == "placeMap"
       || this->stage == "placeTroops")) {
        this->incPlayer();
        this->updateStage();
        if (this->stage == "placeMap") {
            if (this->counter == this->numberOfCountries) {
                this->currentStage->setNextEn(true);
                this->nextState();  //to placeTroops
```

```
            }
        }
        else {
            if (this->counter == (-5 * this->numberOfPlayers + 35)*this->numberOfPlayers) {
                this->currentStage->setNextEn(true);
                this->nextState();  //to draft
            }
        }
    }
}


bool risk::LOAD()
{
    QString fileName = QFileDialog::getOpenFileName(this,
        tr("Load Game"),
        "",
        tr("file (*.dudu);;All Files (*)"));
    if (fileName == "")
        return 0;
    std::ifstream file;
    file.open(fileName.toStdString());
    this->load(file);
    file.close();
    return 1;
}

char risk::get_number_of_player()
{
    return this->numberOfPlayers;
}

void risk::set_number_of_player(char num )
{
    this->numberOfPlayers = num;
}


```

land.h


```
#pragma once
#include "Player.h"
#include "Country.h"
class Land {
public:
    Land();
    Land& operator=(Land&);
    struct landInfo {
        int bonus;
        std::string reason;
    };


    Land(std::vector<std::string>, Land::landInfo);
    Land::landInfo calcTroops(Player&);
    Land::landInfo bonus;

    void save(std::ofstream& file) const;
    void load(std::ifstream& file, std::map<std::string, Player>& playersMap);
public:
    std::vector <Country> countries;
};


Land::landInfo& operator+=(Land::landInfo&, const Land::landInfo&);
```

land.cpp


```cpp
#include "Land.h"

Land::Land()
{
    this->bonus = { 0,"no reason" };
}

Land::Land(std::vector<std::string> countries,Land::landInfo bonus)
{
    this->bonus = bonus;
    int size = countries.size();
    this->countries.reserve(size);

    for (auto iter : countries) {
        this->countries.push_back(Country(iter));
    }
}

Land & Land::operator=(Land &src)
{
    // TODO: insert return statement here
    this->countries = src.countries;
    this->bonus = src.bonus;
    return *this;
}


Land::landInfo Land::calcTroops(Player & player)
{
    landInfo res = { 0,"" };
    for (auto country : this->countries) {
        if (country.getOwner().getName() == player.getName())
            res.bonus++;
    }
    if (res.bonus == this->countries.size()) {
        res.bonus += this->bonus.bonus*3;
        res.reason += this->bonus.reason;
    }
    return res;
}

void Land::save(std::ofstream& file) const {
    for (auto country : this->countries) {
        country.save(file);
    }
}

void Land::load(std::ifstream & file, std::map<std::string, Player>& playersMap)
{

    for (int i = 0; i < this->countries.size();i++) {
        this->countries[i].load(file);
        if (countries[i].getOwner().getName() == "None")
            return;
        Player tmp = playersMap[countries[i].getOwner().getName()];
```

```
        countries[i].setOwner(tmp);
        countries[i].incTroops(0);
    }

}

Land::landInfo & operator+=(Land::landInfo & lhs, const Land::landInfo & rhs)
{
    lhs.reason += "\n" + rhs.reason;
    lhs.bonus += rhs.bonus;
    return lhs;
}

```

country.h


```
#pragma once
#include "Player.h"
#include <QtWidgets/QMainWindow>
#include "ui_my_risk.h"
#include <string.h>
#include <functional>
#include <fstream>
class Country :public QMainWindow {
    Q_OBJECT
public:
    void setButton(QPushButton*);
    static Country* chosen;
    static Country* chosen2;

    static void setUI(Ui::my_riskClass*);
    Country(const std::string, Player player = Player());
    Country(const Country &);
    Country();
    Country& operator=(Country&);

    void setOwner(const Player&);
    void incTroops(int);

    bool validPath(const std::string, std::map<std::string, bool>& seen = std::map<std::string, bool>());
    Player getOwner() const;
    std::string getName() const;
    int getTroops();

    bool isNear(std::string) const;
    void setNehibors(std::vector<Country*>);

    void save(std::ofstream& file) const;
    void load(std::ifstream& file);
public slots:
    void init();
private:
    static Ui::my_riskClass* ui;    //class member
    int troops;
    std::string name;
    Player owner;
    QPushButton* button;
    std::vector<Country*> nehibors;

};
```

country.cpp


```cpp
#include "Country.h"
#include <iostream>
#include <qpushbutton.h>
Country* Country::chosen;
Country* Country::chosen2;
void Country::init()
{
    if (!this->troops) {
        this->ui->textEdit->setText(QString(this->name.c_str()) + QString(" is a FREE country!"));
    }
    else
        this->ui->textEdit->setText(QString("ocupied") + QString(" by player: ") + QString(this->owner.getName().c_str()) + QString(" with: ") + QString(std::to_string(this->troops).c_str()));

    Country::chosen2 = Country::chosen;
    Country::chosen = this;

}


Country::Country(const std::string name, Player player)
{
    this->name = name;
    this->troops = 0;
    this->owner = player;

}

Country::Country(const Country & src)
{
    this->troops = src.troops;
    this->name = src.name;
    this->owner = src.owner;
    this->button = src.button;
}


Country::Country()
{
    //this->button = 0;
}

Country & Country::operator=(Country &src)
{
    // TODO: insert return statement here
    this->troops = src.troops;
    this->name = src.name;
    this->owner = src.owner;
    this->button = src.button;
    return *this;
}

void Country::setButton(QPushButton* button) {
    this->button = button;
```

```cpp
}

void Country::setOwner(const Player& owner)
{
    this->owner = owner;
    QString style = QString("background-color:") + this->owner.getColor().c_str();
    this->button->setStyleSheet(style);
}


void Country::save(std::ofstream& file) const {
    std::regex re("\\s+");
    std::string name = std::regex_replace(this->name, re, "_") ;
    file << name << "\n";
    file << this->troops << "\n";
    file << this->owner.getName() << "\n";
    file << "-----\n";
}

void Country::load(std::ifstream & file)
{
    file >> this->name;
    std::string tmp;
    file >> tmp;
    this->incTroops(std::stoi(tmp));
    std::string ownerName;
    file >> ownerName;
    this->owner.setName(ownerName);
    Country::chosen = this;
    std::string dummy;
    file >> dummy;
}

void Country::incTroops(int add)
{
    this->troops += add;
    if(add !=0)
        this->button->setText(QString(std::to_string(this->troops).c_str()));
}

bool Country::validPath(const std::string dest, std::map<std::string,bool>& seen)
{
    for (int i = 0; i < this->nehibors.size(); i++)
        if (this->nehibors[i]->name == dest)
            return true;

    for (int i = 0; i < this->nehibors.size(); i++)
        if (seen.find(this->nehibors[i]->name) == seen.end()) {
            if (this->nehibors[i]->owner.getName() == this->owner.getName()) {
                seen[this->nehibors[i]->name] = true;
                if (this->nehibors[i]->validPath(dest, seen))
                    return true;
            }
        }
```

```
    return false;
}

Player Country::getOwner() const
{
    return this->owner;
}

std::string Country::getName() const
{
    return std::string(this->name);
}

int Country::getTroops()
{
    return this->troops;
}

bool Country::isNear(std::string name) const
{
    for (int i = 0; i < this->nehibors.size(); i++)
        if (this->nehibors[i]->name == name)
            return true;

    return false;
}

void Country::setNehibors(std::vector<Country*> vec)
{
    this->nehibors.reserve(vec.size());
    for (int i = 0; i < vec.size(); i++) {
        this->nehibors.push_back(vec[i]);
    }
}


Ui::my_riskClass* Country::ui;
void Country::setUI(Ui::my_riskClass* ui) {
    Country::ui = ui;
}

```

Player.h


```
#pragma once
#include "Card.h"
#include <QtWidgets/QMainWindow>
#include <fstream>
#include <regex>
class Player {
public:
    Player(const std::string name = "default", const std::string color = "None", int order = 0, int id = -1);
    Player(const Player&);
    Player& operator=(const Player&);

    void cleanCards(std::vector<Card> cards);

    void addBonusTroops(const int);
    void setBonusTroops(const int);
    int getBonusTroops();

    std::string getName() const;
    void setName(std::string);
    int getId() const;
    void SetOrder(int);
    int GetOrder();
    void show();
    std::string& getColor();
    void save(std::ofstream& file);
    void load(std::ifstream& file);
    Card addCard();
    std::vector<Card> getCards();
private:
    int bonusTroops;
    int order;
    std::vector<Card> cards;
    std::string name;
    std::string color;
    int id;
};

```

Player.cpp


```cpp
#include "Player.h"

Player::Player(const std::string name, const std::string color, int order,int id)
{
    this->name = name;
    this->color = color;
    this->order = order;
    this->bonusTroops = -1;
    this->id = id;
}

Player::Player(const Player &src):cards(src.cards)
{
    this->name = src.name;
    this->color = src.color;
    this->order = src.order;
    this->bonusTroops = src.bonusTroops;
    this->id = src.id;
}

Player & Player::operator=(const Player & src)
{
    this->order = src.order;
    this->color = src.color;
    this->name = src.name;
    this->cards = src.cards;    //todo
    this->bonusTroops = src.bonusTroops;
    this->id = src.id;
    return *this;
}

void Player::cleanCards(std::vector<Card> cards)
{
    for (int i = 0; i < cards.size(); i++) {
        auto it = std::find_if(this->cards.begin(), this->cards.end(), [&](Card x) {return cards[i].getType() == x.getType(); });
        if (it != this->cards.end())
            this->cards.erase(it);
    }
}


void Player::addBonusTroops(const int bonus)
{
    this->bonusTroops += bonus;
}

void Player::setBonusTroops(const int bonus)
{
    this->bonusTroops = bonus;
}

int Player::getBonusTroops()
{
    return this->bonusTroops;
```

```
}


std::string Player::getName() const
{
    return this->name;
}

void Player::setName(std::string name)
{
    this->name = name;
}

int Player::getId() const
{
    return this->id;
}

std::string & Player::getColor()
{
    // TODO: insert return statement here
    return this->color;
}

void Player::save(std::ofstream& file)
{
    std::regex re("\\s+");
    file << std::regex_replace(this->name, re,"_") << "\n";
    file << this->id << "\n";
    file << this->order << "\n";
    file << this->color << "\n";
    file << this->bonusTroops << "\n";
    file << this->cards.size() << "\n";
    for (Card card : cards)
        card.save(file);
    file << "-------\n";

}

void Player::load(std::ifstream & file)
{
    file >> this->name;
    std::string tmp;
    file >> tmp;
    this->id = std::stoi(tmp);
    file >> tmp;
    this->order = std::stoi(tmp);
    file >> this->color;
    file >> tmp;
    this->bonusTroops = std::stoi(tmp);
    int numberOfCards;
    file >> tmp;
    numberOfCards = std::stoi(tmp);
    this->cards.reserve(numberOfCards);
    for (int i = 0; i < numberOfCards; i++) {
        Card tmp;
```

```
        tmp.load(file);
        this->cards.push_back(tmp);
    }
    std::string dummy;
    file >> dummy;
}

Card Player::addCard()
{
    Card tmp;
    tmp.setRand();
    this->cards.push_back(tmp);
    return tmp;
}

std::vector<Card> Player::getCards()
{
    return this->cards;
}


void Player::SetOrder(int order )
{
    this->order = order;
}

int Player::GetOrder()
{
    return this->order;
}

void Player::show()
{
    for (int i = 0; this->cards.size(); ++i)
        this->cards[i].update();
}

```

card.h


```
1  #pragma once
2  #include <QtWidgets/QMainWindow>
3  #include <QWidget>
4  #include <fstream>
5  #include "Qdebug.h"
6  #include "ui_my_risk.h"
7  #define _numberOfValidCards_ 5
8  class Card {
9  public:
10     static void nextBonus();
11     static int  getBonus();
12     enum cardType { Joker, Solider, Cannon, Horse, Default };
13     Card(cardType cardType = cardType::Default);
14     Card(const Card&);
15     Card& operator=(const Card&);
16     bool operator==(const Card&) const;
17     bool operator!=(const Card&) const;
18     void setRand();
19     void setButton(QPushButton*);
20     void update();
21     void save(std::ofstream& file) const;
22     void load(std::ifstream& file);
23     int maxTypes() const;
24     int getType() const;
25     void clean();
26     bool isClicked() const;
27 private:
28     std::string validType[_numberOfValidCards_];
29     cardType type;
30     QPushButton* button;
31     bool clicked;
32     static int bonus1;
33     static int bonus2;
34     static int bonus3;
35 public slots:
36     void click();
37 };
38
```

```
#include "Card.h"
int Card::bonus1;
int Card::bonus2;
int Card::bonus3;
void Card::nextBonus()
{
    Card::bonus1 = Card::bonus2 + Card::bonus3;
    Card::bonus2 = Card::bonus1;
    Card::bonus3 = Card::bonus2;

}

int Card::getBonus()
{
    if (Card::bonus1 == 0) {
        Card::bonus1 = 3;
        Card::bonus2 = 2;
        Card::bonus3 = 1;
    }
    return Card::bonus1;
}

Card::Card(cardType type) : validType { "Joker","Solider","Cannon","Horse","Default" }
{
    this->type = type;
    this->clicked = false;
}

Card::Card(const Card & src) : validType { "Joker","Solider","Cannon","Horse","Default" }
{
    this->type = src.type;
    this->clicked = src.clicked;
}

Card & Card::operator=(const Card & src)
{
    this->type = src.type;
    this->clicked = src.clicked;
    return *this;
}

bool Card::operator==(const Card & rhs) const
{
    if (this->type == Joker || rhs.type == Joker)
        return true;

    return ((int)this->type == (int)rhs.type);
}

bool Card::operator!=(const Card & rhs) const
{
    if (this->type == Joker || rhs.type == Joker)
        return true;

```

```
    return !((int)this->type == (int)rhs.type);
}

void Card::setRand()
{
    int num = rand() % ((_numberOfValidCards_ - 1)*3 + 2);
    if (num != (_numberOfValidCards_ - 1) * 3 + 1) {    //chance of 1 to (_numberOfValidCards_ - 1)*3 and every other card chance of 1 to (_numberOfValidCards_ - 1)
        num = num % 3;
    }
    else {
        num = 0;
    }
    this->type = Card::cardType(num);   //= { "Joker","Solider","Cannon","Horse" };
}

void Card::setButton(QPushButton * button)
{
    this->button = button;

}

void Card::update()
{
    //QPixmap pic("Resources/cards/card"+ QString(this->validType[this->type].c_str()) + ".PNG");
    QString pic = "border-image:url(:/new/Resources/cards/card" + QString(this->validType[this->type].c_str()) + ".PNG);";
    //this->button->setIcon()
    this->button->setStyleSheet(pic);

    this->button->show();

}

void Card::click()
{
    if (this->type == Card::Default)
        return;

    QString pic;
    if (this->clicked) {
        this->clicked = false;
        pic = "border-image:url(:/new/Resources/cards/card" + QString(this->validType[this->type].c_str()) + ".PNG);";
    }
    else {
        this->clicked = true;
        pic = "border-image:url(:/new/Resources/cards/card" + QString(this->validType[this->type].c_str()) + "Shadow.PNG);";
    }

    //this->button->setIcon()
    this->button->setStyleSheet(pic);
```

```cpp
}

void Card::save(std::ofstream & file) const
{
    file << this->validType[this->type] << "\n";
}

void Card::load(std::ifstream & file)
{
    std::string tmp;
    file >> tmp;
    if (tmp == "Solider")
        this->type = Solider;
    else if (tmp == "Cannon")
        this->type = Cannon;
    else if (tmp == "Horse")
        this->type = Horse;
    else if (tmp == "Joker")
        this->type = Joker;
    else
        this->type = Default;
}

int Card::maxTypes() const
{
    return _numberOfValidCards_;
}

int Card::getType() const
{
    return int(this->type);
}

void Card::clean()
{
    QString pic = "border-image:url(:/new/Resources/cards/card" + QString (this->validType[Card::Default].c_str()) + ".PNG);";
    this->button->setStyleSheet(pic);
    this->clicked = false;
    this->type = Default;
    this->button->show();
}

bool Card::isClicked() const
{
    return this->clicked;
}

```

Cube.h


```
#pragma once
#include <QWidget>
#include "ui_menu.h"
#include <string>
#include <iostream>

class Cube {
private:
    QLabel* label;
    static std::string path;
    int rand_value;
public:
    Cube();
    void choosePic(int);
    void setValue(int);
    int getValue() const;

    static void setPath(std::string);

    void setLabel(QLabel*);
    void setPic(std::string);
    void random();
    void hide();
    void show();
    bool operator<(const Cube&) const;
    bool operator>(const Cube&) const;
    bool operator==(const Cube&) const;

};
```

cube.cpp


```
#include "Cube.h"
#include "Vector.h"
std::string Cube::path;
void Cube::choosePic(int num)
{

    QPixmap pic(QString(this->path.c_str()) + QString(num + '0') + ".png");//
      ("Resources/cubes/1.png");
    label->setPixmap(pic);
    label->show();
}
void Cube::setValue(int rand_val)
{
    this->rand_value = rand_val;
}
int Cube::getValue() const
{
    return this->rand_value;
}
Cube::Cube()
{
    this->rand_value = -1;
}

void Cube::setPath(std::string path)
{
    Cube::path = path;
}


void Cube::setLabel(QLabel * label)
{
    this->label = label;
    this->label->setVisible(true);
}

void Cube::setPic(std::string str)
{

    QPixmap pic(str.c_str());//("Resources/cubes/1.png");
    label->setPixmap(pic);

    label->show();
}

void Cube::random()
{
    this->rand_value = std::rand() % 6 + 1;
    this->choosePic(this->rand_value);
}

void Cube::hide()
{
    {
        this->label->setVisible(false);
    }
```

```
}

void Cube::show()
{
    this->label->setVisible(true);
}

bool Cube::operator<(const Cube & rhs) const
{
    return this->rand_value < rhs.rand_value;
}

bool Cube::operator>(const Cube & rhs) const
{
    return this->rand_value > rhs.rand_value;
}

bool Cube::operator==(const Cube & rhs) const
{
    return this->rand_value == rhs.rand_value;
}

```

Stage.h


```
#pragma once
#include <QtWidgets/QMainWindow>
#include "ui_my_risk.h"
#include "Player.h"
#include "Country.h"
#include "Cube.h"
#include "Move.h"
class Stage {
public:
    Stage(Ui::my_riskClass&, Player& player, Move& moveUi);
    virtual void preAction()=0;
    virtual void action()=0;
    virtual void afterAction()=0;
    virtual void plus(int sel = 0);
    virtual void minus(int sel = 0);
    void showTroops();
    bool getNexEn();
    void setNextEn(const bool en);
    bool getEnterEn();
    void setEnterEn(const bool en);
    void setPlayer(Player&);
protected:
    int getBonusTroops(int sel = 0);
    bool isMine(int sel = 0);

protected:
    Ui::my_riskClass& ui;
    Move& moveUi;
    Player& player;
    bool nextEn;
    bool EnterEn;



};


```

stage.cpp


```cpp
#include "Stage.h"

Stage::Stage(Ui::my_riskClass& ui,Player& player, Move& moveUi): ui(ui),player (player),moveUi(moveUi)
{
    this->nextEn = false;
    this->EnterEn = true;
}

bool Stage::getNexEn()
{
    return this->nextEn;
}

void Stage::setNextEn(const bool en)
{
    this->nextEn = en;
}

bool Stage::getEnterEn()
{
    return this->EnterEn;
}

void Stage::setEnterEn(const bool en)
{
    this->EnterEn = en;
}

void Stage::setPlayer(Player & player)
{
    this->player = player;
}

void Stage::showTroops()
{
    int bonus = this->player.getBonusTroops();
    QString text = std::to_string(bonus).c_str();
    this->ui.troopsP->setText(text);
}
int Stage::getBonusTroops(int sel)
{
    if (sel == 0) {
        std::string text = this->ui.troopsP->toPlainText().toStdString();
        return std::stoi(text);
    }
    else if (sel == 1) {
        std::string text = this->ui.troopsP_2->toPlainText().toStdString();
        return std::stoi(text);
    }
    return -1;
}

bool Stage::isMine(int sel)
{

```

```
    if (sel == 0) {
        if (Country::chosen->getOwner().getName() != this->player.getName()) {
            this->ui.textEdit->setText("not your country");
            return false;
        }
        return true;
    }
    else {
        if (Country::chosen2->getOwner().getName() != this->player.getName()) {
            this->ui.textEdit->setText("not your country");
            return false;
        }
        return true;
    }
}

void Stage::plus(int sel)
{

}

void Stage::minus(int sel)
{
    bool force = false;
    if (sel >= 10) {
        force = 1;
        sel %= 10;
    }
    if (sel == 0) {
        int number = this->getBonusTroops() - 1;
        if (number > 0 || force)
            this->ui.troopsP->setText(std::to_string(number).c_str());
        else
            this->ui.textEdit->setText("I see you got NA education ...");
    }
    else if (sel == 1) {
        int number = this->getBonusTroops(1) - 1;
        if (number > 0 || force)
            this->ui.troopsP_2->setText(std::to_string(number).c_str());
        else
            this->ui.textEdit->setText("I see you got NA education ...");
    }
}
```

PlaceMap.h


```
#pragma once
#include "Stage.h"
#include "Land.h"
#include "Player.h"
#include "Country.h"

class PlaceMap : public Stage {
public:
    PlaceMap(Ui::my_riskClass&, Player& player, Move& moveUi, int& counter);

    void action();
    void preAction();
    void afterAction();
private:
    int& counter;
};
```

PlaceMap.cpp


```cpp
#include "PlaceMap.h"

PlaceMap::PlaceMap(Ui::my_riskClass & ui, Player & player,Move& moveUi , int& counter) : Stage(ui, player, moveUi), counter(counter)
{
}

void PlaceMap::action()
{
    if (this->isMine() && Country::chosen->getTroops()) {
        this->ui.textEdit->setText(QString("At this stage you need to fill-up the map .."));
        this->EnterEn = false;
        return;
    }

    if (Country::chosen->getTroops()) {
        this->ui.textEdit->setText(QString("I see you got NA education .."));
        this->EnterEn = false;
        return;
    }
    this->EnterEn = true;
    Country::chosen->setOwner(this->player);
    Country::chosen->incTroops(1);
    this->counter++;
    this->ui.textEdit->setText(QString("initial Country: ") + QString(Country::chosen->getName().c_str()) + QString(" by player: ") + QString(this->player.getName().c_str()));
    this->player.addBonusTroops(-1);

}

void PlaceMap::preAction()
{
    this->showTroops();
}

void PlaceMap::afterAction()
{
    this->showTroops();
}

```

PlaceTropps.h


```
1  #pragma once
2  #include "Stage.h"
3  #include "Land.h"
4  #include "Player.h"
5  #include "Country.h"
6
7  class PlaceTroops : public Stage {
8  public:
9      PlaceTroops(Ui::my_riskClass&, Player& player, Move& moveUi,int& counter);
10
11     void action();
12     void preAction();
13     void afterAction();
14
15     void showInfo();
16 private:
17     int& counter;
18
19 };
```

PlaceTroops.cpp


```cpp
#include "PlaceTroops.h"

PlaceTroops::PlaceTroops(Ui::my_riskClass & ui, Player & player, Move & moveUi,int& counter) : Stage(ui, player, moveUi), counter(counter)
{
}

void PlaceTroops::action()
{
    if (!this->isMine())
    {
        this->ui.textEdit->setText(QString(Country::chosen->getName().c_str()) + QString(" isn't yours"));
        this->EnterEn = false;
        return;
    }
    this->EnterEn = true;
    this->counter++;
    this->ui.textEdit->setText(QString("add one troop at") + QString(Country::chosen->getName().c_str()) + QString(" to ") + QString(this->player.getName().c_str()));
    Country::chosen->incTroops(1);
    this->player.addBonusTroops(-1);
}


void PlaceTroops::preAction()
{
    this->showInfo();
    this->showTroops();
}

void PlaceTroops::afterAction()
{

}

void PlaceTroops::showInfo()
{
    QString picPath = QString("Resources/players/") + QString(this->player.getColor().c_str()) + ".png";
    QPixmap pic(picPath);//("Resources/cubes/1.png");
    this->ui.colorP->setPixmap(pic);
    this->ui.colorP->show();
}

```

Draft.h


```
#pragma once
#include "Stage.h"
#include "Land.h"
#include "Player.h"
#include "Country.h"
class Draft : public Stage {
public:
    Draft(Ui::my_riskClass&, Player& player, Move& moveUi, std::vector<Card>&
      cards,int _maxCardsPerPlayer);

    void setLands(std::vector<Land*> lands);
    void action();
    void preAction();
    void afterAction();
    void showBonusTroops(std::string);
    Land::landInfo calcTroops();
    void plus(int sel = 0);
    void minus(int sel = 0);
    void convert();
private:
    int newCard;
    std::vector<Card>& cards;
    std::vector<Land*> lands;
    int _maxCardsPerPlayer;

};
```

```cpp
#include "Draft.h"

Draft::Draft(Ui::my_riskClass & ui, Player& player, Move& moveUi, std::vector<Card>& cards, int _maxCardsPerPlayer_) : Stage(ui, player, moveUi), cards(cards)
{
    this->cards = cards;
    this->_maxCardsPerPlayer = _maxCardsPerPlayer_;
    this->ui.textEdit->setText(QString("welcome to stage draft you need to place all your troops in this stage:"));
}

void Draft::setLands(std::vector<Land*> lands)
{
    this->lands = lands;
}
void Draft::action()
{
    if (!this->isMine() || this->EnterEn == false)

        return;

    this->ui.textEdit->setText(
        "add" + QString(std::to_string(this->getBonusTroops()).c_str()) + " number of troops at: "
        + Country::chosen->getName().c_str() + " to: " + this->player.getName().c_str());
    Country::chosen->incTroops(this->getBonusTroops());
    this->player.addBonusTroops((-1)*this->getBonusTroops());
    this->showTroops();
    if (this->player.getBonusTroops() == 0) {
        this->nextEn = true;
        this->EnterEn = false;
    }
}

void Draft::preAction()
{
    Land::landInfo info = this->calcTroops();
    if (info.bonus == 0)
    {
        this->nextEn = true;
    }
    this->player.setBonusTroops(info.bonus);
    this->showBonusTroops(info.reason);
    std::vector<Card> cards = this->player.getCards();
    for (int i = 0; i < 5; i++) //todo const
        if (i < cards.size())
            this->cards[i] = cards[i];
        else
            this->cards[i] = Card();

    for (int i = 0; i < _maxCardsPerPlayer; i++)
        this->cards[i].update();
}

```

```
52  void Draft::afterAction()
53  {
54      if (this->player.getCards().size() == _maxCardsPerPlayer) {
55          this->ui.textEdit->setText("You must convert cards to troops!");
56          this->nextEn = false;
57      }
58  }
59
60  void Draft::showBonusTroops(std::string reason)
61  {
62      this->ui.troopsP->setText(std::to_string(this->player.getBonusTroops
        ()).c_str());
63      if (reason == "")
64          this->ui.textEdit->append("no land is yours!");
65      else
66          this->ui.textEdit->append(reason.c_str());
67  }
68
69  Land::landInfo Draft::calcTroops()
70  {
71      Land::landInfo info = { 0,"" };
72      for (std::vector<Land*>::iterator it = lands.begin(); it != lands.end(); +
        +it)
73          info += (*it)->calcTroops(this->player);
74
75      if (info.bonus == 0) {
76          info.bonus = -1;
77          return info;
78      }
79      info.bonus /= 3;
80      if (info.bonus < 3)
81          info.bonus = 3;
82      return info;
83  }
84
85  void Draft::plus(int sel)
86  {
87      if (sel == 0) {
88          int number = this->getBonusTroops() + 1;
89          if (number <= this->player.getBonusTroops())
90              this->ui.troopsP->setText(std::to_string(number).c_str());
91          else
92              this->ui.textEdit->setText("too many troops! sorry ..");
93      }
94      else if (sel == 1) {
95          int number = this->getBonusTroops(1) + 1;
96          if (number <= this->player.getBonusTroops())
97              this->ui.troopsP_2->setText(std::to_string(number).c_str());
98          else
99              this->ui.textEdit->setText("too many troops! sorry ..");
100     }
101 }
102
103 void Draft::minus(int sel)
104 {
105     bool force = false;
```

```
    if (sel >= 10) {
        force = 1;
        sel %= 10;
    }
    if (sel == 0) {
        int number = this->getBonusTroops() - 1;
        if (number > 0 || force)
            this->ui.troopsP->setText(std::to_string(number).c_str());
        else
            this->ui.textEdit->setText("I see you got NA education ...");
    }
    else if (sel == 1) {
        int number = this->getBonusTroops(1) - 1;
        if (number > 0 || force)
            this->ui.troopsP_2->setText(std::to_string(number).c_str());
        else
            this->ui.textEdit->setText("I see you got NA education ...");
    }
}

void Draft::convert()
{
    std::vector<Card> cards;
    for (int i = 0; i < this->cards.size(); i++) {  //3 same type
        if (this->cards[i].getType() != Card::Default && this->cards[i].isClicked()) {
            cards.push_back(this->cards[i]);
        }
    }
    if (cards.size() != 3) {
        this->ui.textEdit->setText("no matching cards for convert! choose 3 valid cards! ");
        return;
    }

    if (cards[0] == cards[1] && cards[0] == cards[2]) {
        //check tripple of the same type / joker type.
        for (int i = 0; i < cards.size(); i++) {
            auto it = std::find_if(this->cards.begin(), this->cards.end(), [&](Card x) {return cards[i].getType() == x.getType(); });
            if (it != this->cards.end())
                it->clean();
        }
        this->player.cleanCards(cards);
        int bonus = Card::getBonus();
        this->player.addBonusTroops(bonus);
        Card::nextBonus();
        this->ui.troopsP->setText(std::to_string(this->player.getBonusTroops()).c_str());
        this->ui.textEdit->setText("N1! get bonus of " +
            QString(std::to_string(bonus).c_str())+
            " troops!");
        this->EnterEn = true;
    }
    else if(cards[0] != cards[1] && cards[0] != cards[2] && cards[1] != cards[2]){
```

```
        //check tripple of diffrent types.
        for (int i = 0; i < cards.size(); i++) {
            auto it = std::find_if(this->cards.begin(), this->cards.end(), [&] (Card x) {return cards[i].getType() == x.getType(); });
            if (it != this->cards.end())
                it->clean();
        }
        this->player.cleanCards(cards);
        int bonus = Card::getBonus();
        this->player.addBonusTroops(bonus);
        Card::nextBonus();
        this->ui.troopsP->setText(std::to_string(bonus).c_str());
        this->EnterEn = true;
    }
    else {
        this->ui.textEdit->setText("no matching cards for convert! choose 3 valid cards! ");
        return;
    }



}

```

```cpp
#include "Attack.h"

Attack::Attack(Ui::my_riskClass & ui, Player& player, Move& moveUi, Card* card) : Stage(ui,player,moveUi)
{
    this->newCard = false;
    this->card = card;
    this->setNextEn(true);
    this->ui.TROOPS_2->setVisible(true);
    this->boom = nullptr;
}

void Attack::action()
{


    if (this->getBonusTroops() == 0 || this->getBonusTroops(1) == 0) {
        this->ui.textEdit->setText("I see you got NA education .. choose number of troops!");
        return;
    }


    //chosen2 attaker
    //chosen defender
    if (!this->isMine(1))//attacker is not me.
        return;
    if (this->isMine()) {   //country to attack is mine.
        this->ui.textEdit->setText("I see you got NA education .. you cannot attack yourself!");
        return;
    }
    if (!Country::chosen2->isNear(Country::chosen->getName())) {    //cannot attack un-near countries.
        this->ui.textEdit->setText("they are not near!");
        return;
    }


    //troops and countries are valid
    //attack logic --
    //make cubes according to troops.
    //random values.
    std::vector<Cube> attacker;
    std::vector<Cube> defender;

    attacker.reserve(this->getBonusTroops());
    for (int i = 0; i < this->getBonusTroops(); i++)
        attacker.push_back(Cube());
    attacker[0].setLabel(this->ui.cubeP_1);
    if (attacker.size() > 1)
        attacker[1].setLabel(this->ui.cubeP_2);
    if (attacker.size() > 2)
        attacker[2].setLabel(this->ui.cubeP_3);

    defender.reserve(this->getBonusTroops(1));
```

```
    for (int i = 0; i < this->getBonusTroops(1); i++)
        defender.push_back(Cube());
    defender[0].setLabel(this->ui.cubeP2_1);
    if (defender.size() > 1)
        defender[1].setLabel(this->ui.cubeP2_2);


    for (int i = 0; i < attacker.size(); i++)
        attacker[i].random();
    for (int i = 0; i < defender.size(); i++)
        defender[i].random();

    //std::sort(attacker.begin(), attacker.end(), [](const Cube& x, const Cube& y) {return x.get_rand_value() > y.get_rand_value(); });
    //std::sort(defender.begin(), defender.end(), [](const Cube& x, const Cube& y) {return x.get_rand_value() > y.get_rand_value(); });
    std::sort(attacker.begin(), attacker.end());//from small to big
    std::sort(defender.begin(), defender.end());
    int cnt = 0;
    if (std::min(attacker.size(), defender.size()) == 1)    //is 1 troop do the loop only once.
        cnt = 1;
    do {
        cnt++;
        if (attacker.back() > defender.back()) {    //attaker have to biggest cube.
            auto it = std::find_if(attacker.begin(), attacker.end(), [&](const Cube& x) {return x > defender.back(); });  //go from small to big.
            //del both cubes.
            attacker.erase(it);
            defender.pop_back();

            Country::chosen->incTroops(-1);     //erase from defender!
            this->minus(11);
        }
        else {                                  //else
            auto it = std::find_if(defender.begin(), defender.end(), [&](const Cube& x) {return (!(attacker.back() > x)); }); //go from small to big.
            //del both cubes.
            defender.erase(it);
            attacker.pop_back();

            Country::chosen2->incTroops(-1);        //erase from attaker!
            this->minus(10);
        }
    } while (this->getBonusTroops() && this->getBonusTroops(1) && cnt != 2);   //end if 2 loops or troops is empty.


    if (Country::chosen->getTroops())
        return;
    this->boom= new QMediaPlayer();
     this->boom->setMedia(QUrl("qrc:/new/Resources/Explosion.wav"));
    this->boom->play();
```

```
    this->newCard = true;
    Country::chosen->setOwner(this->player);
    this->moveUi.setTo(Country::chosen);    //send to defender
    this->moveUi.setFrom(Country::chosen2); //send From attaker
    this->moveUi.setMax(Country::chosen2->getTroops() - 1);
    this->moveUi.setMin(this->getBonusTroops());
    this->moveUi.setFocus();
    this->moveUi.show();
}

void Attack::preAction()
{
    this->ui.textEdit->setText("Please choose a country to attack and the number of troops to send");
    this->ui.troopsP->setText("1");
    this->ui.troopsP_2->setText("1");
}

void Attack::afterAction()
{
    if (this->newCard) {    //if new card update gui and player.
        this->newCard = false;
        Card tmp = this->player.addCard();
        *this->card = tmp;
        this->card->update();
    }
    this->ui.cubeP_1->hide();
    this->ui.cubeP_2->hide();
    this->ui.cubeP_3->hide();
    this->ui.cubeP2_1->hide();
    this->ui.cubeP2_2->hide();
    this->ui.cubeP2_3->hide();
    if (this->boom)
    {
        delete boom;
    }
    this->ui.TROOPS_2->setVisible(false);
}

void Attack::plus(int sel)
{
    if (sel == 0) {
        int number = this->getBonusTroops() + 1;
            if (number > 3 || number >= Country::chosen2->getTroops())
                this->ui.textEdit->setText("too many troops! sorry ..");
            else
                this->ui.troopsP->setText(std::to_string(number).c_str());
    }
    else if (sel == 1) {
        int number = this->getBonusTroops(1) + 1;

            if (number > 2 || number > Country::chosen->getTroops())
                this->ui.textEdit->setText("too many troops! sorry ..");
            else
                this->ui.troopsP_2->setText(std::to_string(number).c_str());
    }
```

```
}

void Attack::minus(int sel)
{
    bool force = false;
    if (sel >= 10) {
        force = 1;
        sel %= 10;
    }
    if (sel == 0) {
        int number = this->getBonusTroops() - 1;
            if (number > 0 || force)
                this->ui.troopsP->setText(std::to_string(number).c_str());
            else
                this->ui.textEdit->setText("I see you got NA education ...");
    }
    else if (sel == 1) {
        int number = this->getBonusTroops(1) - 1;
            if (number > 0 || force)
                this->ui.troopsP_2->setText(std::to_string(number).c_str());
            else
                this->ui.textEdit->setText("I see you got NA education ...");
    }
}

```

Reinforcement.h


```
1  #pragma once
2  #include "Stage.h"
3  #include "Land.h"
4  #include "Player.h"
5  #include "Country.h"
6
7  class Reinforcement : public Stage {
8  public:
9      Reinforcement(Ui::my_riskClass& ui, Player& player, Move& moveUi);
10
11     void action();
12     void preAction();
13     void afterAction();
14
15
16
17 };
```

Reinforcement.cpp


```cpp
#include "Reinforcement.h"

Reinforcement::Reinforcement(Ui::my_riskClass & ui, Player & player, Move & moveUi) : Stage(ui, player, moveUi)
{
    this->setNextEn(true);
    this->ui.TROOPS->setVisible(false);
}

void Reinforcement::action()
{
    if (Country::chosen->getOwner().getName() != Country::chosen2->getOwner().getName() || !this->isMine()) {
        this->ui.textEdit->setText("Please choose 2 owned countries which are connected!");
        return;
    }
    if (!Country::chosen2->validPath(Country::chosen->getName())) {
        this->ui.textEdit->setText("Those countries are not connected!");
        return;
    }
    if (Country::chosen2->getTroops() == 1)
    {
        this->ui.textEdit->setText("you cant move troops from country that have inly 1 soilder!");
        return;
    }
    //pop up

    this->moveUi.setTo(Country::chosen);    //send to
    this->moveUi.setFrom(Country::chosen2); //send From
    this->moveUi.setMax(Country::chosen2->getTroops() - 1);
    this->moveUi.setMin(1);
    this->moveUi.setFocus();
    this->moveUi.show();
}

void Reinforcement::preAction()
{

}

void Reinforcement::afterAction()
{
    this->ui.TROOPS->setVisible(true);
}

```

Vector.h


```
#pragma once
#include "iostream"
template <typename T>
class  Vector {
public:
    Vector();
    typedef T * iterator;
    Vector(const Vector<T>& v);
    ~Vector();
    //geters
    unsigned int capacity() const;
    unsigned int size() const;
    bool empty() const;
    T& front();
    T& back();
    iterator begin();
    iterator end();
    void push_back(const T& value);
    void pop_back();
    void reserve(unsigned int capacity);
    T & operator[](unsigned int index);
    T &  operator[](unsigned int index)const ;
    void clear();
    Vector<T> & operator = (const Vector<T> &);
private:
    unsigned int size_;
    unsigned int capacity_;
    T* buffer;
};

template<class T>
Vector<T>::Vector() {
    capacity_ = 0;
    size_ = 0;
    buffer = nullptr;
}

template<class T>
Vector<T>::~Vector() {
    if (this->capacity_)
        delete[] buffer;
}
template<class T>
Vector<T>::Vector(const Vector<T> & v) {
    size_ = v.size_;
    this->capacity_ = v.capacity_;
    if (this->buffer)
    {
        delete[] buffer;
    }
    buffer = new T[this->capacity_];
    for (unsigned int i = 0; i < size_; i++)
        buffer[i] = v.buffer[i];
}
template<class T>
void Vector<T>::reserve(unsigned int capacity) {
```

```
    T * newBuffer = new T[capacity];
    for (int i = 0; i < this->size_; i++)
        newBuffer[i] = buffer[i];
    this->capacity_ = capacity;
    delete[] buffer;
    buffer = newBuffer;
}

template<class T>
void Vector<T>::push_back(const T & v) {
    if (this->size_ >= this->capacity_) {
        if (!this->capacity_)
        {
            this->capacity_ = 1;
        }
        reserve(2 * capacity_);
    }
    buffer[size_++] = v;
}

template<class T>
void Vector<T>::pop_back() {// at the end we delete vector at size capacity
    size_--;
}
template<class T>
T& Vector<T>::front() {
    return buffer[0];
}

template<class T>
T& Vector<T>::back() {
    return buffer[this->size_ - 1];
}

template<class T>
typename Vector<T>::iterator Vector<T>::end() {
    return buffer + size_;
}

template<class T>
typename Vector<T>::iterator Vector<T>::begin() {
    return buffer;
}
template <class T>
bool Vector<T>::empty() const {
    return size_ == 0;
}

template<class T>
Vector<T>& Vector<T>::operator = (const Vector<T> & v) {
    if (buffer)
        delete[] buffer;
    this->size_ = v.size_;
    this->capacity_ = v.capacity_;
    buffer = new T[this->capacity_];
    for (unsigned int i = 0; i < this->size_; i++)
```

```
        buffer[i] = v.buffer[i];
    return *this;
}
template<class T>
unsigned int Vector<T>::size() const {
    return size_;
}

template<class T>
T& Vector<T>::operator[](unsigned int index) {
    return buffer[index];
}
template<class T>
 T& Vector<T>::operator[](unsigned int index)const  {
    return buffer[index];
}

template<class T>
void Vector<T>::clear() {// at the end we delete vector at size capacity
    if (buffer != nullptr)
        delete[] buffer;
    this->size_ = 0;
    this->capacity_ = 0;
}

template<class T>
unsigned int Vector<T>::capacity()const {
    return this->capacity_;
}

```

PlayerStats.h


```
#pragma once
#include "Player.h"
#include "PlayerStatWrap.h"
#include <fstream>
#include "Vector.h"
class PlayerStats {
public:
    PlayerStats();

    template <typename T>
    struct Info {
        std::string owner;
        T value;
    };

    PlayerStats::Info<int> getMaxWins();
    PlayerStats::Info<int> getMaxLooses();
    PlayerStats::Info<int> getMaxTimePlayed();
    PlayerStats::Info<int> getMaxGamesPlayed();
    const Vector<PlayerStatWrap>& get_players();
    std::string getAvgTime();
    void loadStats(std::ifstream& file);
    void saveStats(std::ofstream& file);
    void addToStats(const PlayerStatWrap&);
    void clear();
    int size();
private:
    Vector<PlayerStatWrap> players;
    //std::vector<PlayerStatWrap> players;
    PlayerStats::Info<int> getMaxInt(int(*)(const PlayerStatWrap));

};


```

PlayerStats.cpp


```cpp
#include "PlayerStats.h"


PlayerStats::PlayerStats()
{

}

PlayerStats::Info<int> PlayerStats::getMaxWins()
{
    Info<int> res = { "" ,0 };
    for (int i = 0; i < this->players.size(); ++i) {
        if (this->players[i].get_wins() >= res.value) {
            res.value = this->players[i].get_wins();
            res.owner = this->players[i].get_name();
        }
    }
    return res;
}

void PlayerStats::loadStats(std::ifstream & file)
{
    while (file.eof() == false) {
        PlayerStatWrap res;
        res.loadStats(file);
        this->addToStats(res);
    }

}

void PlayerStats::saveStats(std::ofstream & file)
{
    for (int i = 0; i < this->players.size(); i++) {
        this->players[i].saveStats(file);
        if (i + 1 != this->players.size())
            file << "\n";
    }
}


void PlayerStats::addToStats(const PlayerStatWrap& playerWrap)
{
    for (int i = 0; i < this->players.size(); i++) {
        if (this->players[i].get_name() == playerWrap.get_name()) {
            this->players[i] = this->players[i] + playerWrap;
            return;
        }
    }
    this->players.push_back(playerWrap);
}

void PlayerStats::clear()
{
    this->players.clear();
}

```

```cpp

int PlayerStats::size()
{
    return this->players.size();
}

PlayerStats::Info<int> PlayerStats::getMaxInt(int(*)(const PlayerStatWrap))
{
    return PlayerStats::Info<int>();
}

PlayerStats::Info<int> PlayerStats::getMaxLooses()
{
    Info<int> res = { "" ,0 };
    for (int i = 0; i < this->players.size(); ++i) {
        if (this->players[i].get_looses() >= res.value) {
            res.value = this->players[i].get_looses();
            res.owner = this->players[i].get_name();
        }
    }
    return res;
}

PlayerStats::Info<int> PlayerStats::getMaxTimePlayed()
{
    Info<int> res = { "" ,0 };
    for (int i = 0; i < this->players.size(); ++i) {
        if (this->players[i].get_timePlayed() >= res.value) {
            res.value = this->players[i].get_timePlayed();
            res.owner = this->players[i].get_name();
        }
    }
    return res;
}

PlayerStats::Info<int> PlayerStats::getMaxGamesPlayed()
{
    Info<int> res = { "" ,0 };
    for (int i = 0; i < this->players.size(); ++i) {
        if (this->players[i].get_gamesPlayed() >= res.value) {
            res.value = this->players[i].get_gamesPlayed();
            res.owner = this->players[i].get_name();
        }
    }
    return res;
}

const Vector<PlayerStatWrap>& PlayerStats::get_players()
{
    return this->players;
}


std::string PlayerStats::getAvgTime()
{
    int time = 0;
```

```
    for (int i = 0; i < this->players.size(); ++i) {
            time += this->players[i].get_timePlayed();
    }
    return "the avg of time played in min is: " + std::to_string(time/
      (60*this->players.size()));
}
```

PlayerStatsWarp.h


```
1  #pragma once
2  #include "Player.h"
3  #include <chrono>
4  #include <fstream>
5  class PlayerStatWrap {
6  public:
7      PlayerStatWrap();
8      PlayerStatWrap(const PlayerStatWrap& src);
9      PlayerStatWrap(const Player& player,int win, int elapsed_seconds);
10     void loadStats(std::ifstream& file);
11     void saveStats(std::ofstream& file);
12     std::string get_name() const;
13     int get_timePlayed() const;
14     int get_gamesPlayed() const;
15     int get_wins() const;
16     int get_looses() const;
17     int get_id() const;
18     PlayerStatWrap& operator+(const PlayerStatWrap& src);
19     PlayerStatWrap& operator=(const PlayerStatWrap& src);
20 private:
21     int timePlayed;
22     int gamesPlayed;
23     int wins;
24     int looses;
25     int id; //todo
26     std::string name;
27     //int maxTroops;
28     //int maxCountries;
29
30 };
```

PlayerStatsWarp.cpp


```
#include "PlayerStatWrap.h"
//getline(fileHandle, org, ' ');
PlayerStatWrap::PlayerStatWrap()
{
    this->timePlayed = 0;
    this->gamesPlayed = 1;
    this->wins = 0;
    this->looses = 0;
    this->id = 0;
}

PlayerStatWrap::PlayerStatWrap(const PlayerStatWrap & src)
{
    this->name = src.name;
    this->wins = src.wins;
    this->looses = src.looses;
    this->timePlayed = src.timePlayed;
    this->gamesPlayed = src.gamesPlayed;
    this->id = src.id;
}

PlayerStatWrap::PlayerStatWrap(const Player& player, int win, int elapsed_seconds)
{
    this->name = player.getName();
    if (win == 0) {
        this->wins = 0;
        this->looses = 0;
    }
    else if (win == 1) {
        this->wins = 1;
        this->looses = 0;
    }
    else if (win == -1) {
        this->wins = 0;
        this->looses = 1;
    }
    this->timePlayed = elapsed_seconds;
    this->gamesPlayed = 1;
    this->id = player.getId();
}

void PlayerStatWrap::loadStats(std::ifstream & file)
{
    std::string line;
    std::getline(file, line, ',');
    this->name = line;

    std::getline(file, line, ',');
    this->id = std::stoi(line);

    std::getline(file, line, ',');
    this->wins = std::stoi(line);

    std::getline(file, line, ',');
    this->looses = std::stoi(line);
```

```

    std::getline(file, line, ',');
    this->timePlayed = std::stoi(line);

    std::getline(file, line);
    this->gamesPlayed = std::stoi(line);

}

void PlayerStatWrap::saveStats(std::ofstream & file)
{
    file << this->name << ","
        << this->id << ","
        << this->wins << ","
        << this->looses << ","
        << this->timePlayed << ","
        << this->gamesPlayed;
}
int PlayerStatWrap::get_timePlayed() const
{
    return this->timePlayed;
}

int PlayerStatWrap::get_gamesPlayed() const
{
    return this->gamesPlayed;
}

int PlayerStatWrap::get_wins() const
{
    return this->wins;
}

int PlayerStatWrap::get_looses() const
{
    return this->looses;
}

int PlayerStatWrap::get_id() const
{
    return this->id;
}

std::string PlayerStatWrap::get_name() const
{
    return this->name;
}

PlayerStatWrap & PlayerStatWrap::operator+(const PlayerStatWrap & src)
{
    this->wins += src.wins;
    this->looses += src.looses;
    this->timePlayed += src.timePlayed;
    this->gamesPlayed += src.gamesPlayed;
    return *this;
}
```

```

PlayerStatWrap & PlayerStatWrap::operator=(const PlayerStatWrap & src)
{
    this->name = src.name;
    this->wins = src.wins;
    this->looses = src.looses;
    this->timePlayed = src.timePlayed;
    this->gamesPlayed = src.gamesPlayed;
    this->id = src.id;
    return *this;
}

```

Mess.h


```
1  #pragma once
2  #include <QWidget>
3  #include "ui_Mess.h"
4  //#include "ui_Mess.h"
5
6  class Mess : public QWidget {
7      Q_OBJECT
8
9  public:
10     Mess(QWidget * parent = Q_NULLPTR);
11     ~Mess();
12     void setMess(std::string str);
13 private:
14     Ui::mess_gui ui;
15
16
17 };
```

Mess.cpp


```
1  #include "Mess.h"
2
3  Mess::Mess(QWidget *parent) :QWidget(parent)
4  {
5      ui.setupUi(this);
6  }
7  Mess::~Mess()
8  {
9
10 }
11
12 void Mess::setMess(std::string str)
13 {
14     this->ui.textEdit->setText(str.c_str());
15     this->ui.textEdit->show();
16 }
17
```

StatsGuiMain.h


```
1  #pragma once
2  #include "ui_stats.h"
3  #include "PlayerStats.h"
4  class statsGuiMain :public QWidget {
5      Q_OBJECT
6
7  public:
8      statsGuiMain(QWidget * parent = Q_NULLPTR);
9      ~statsGuiMain();
10
11     void setStats(PlayerStats* stats);
12     void clear();
13 private:
14     Ui::statsGui ui;
15     PlayerStats* stats;
16 private slots:
17     void spin(int);
18     void player_stats(int);
19 };
```

```cpp
#include "statsGuiMain.h"

statsGuiMain::statsGuiMain(QWidget * parent) : QWidget(parent)
{
    ui.setupUi(this);
}

statsGuiMain::~statsGuiMain()
{

}

void statsGuiMain::setStats(PlayerStats * stats)
{
    this->stats = stats;
    QString text;
    text = (QString)this->stats->getAvgTime().c_str() + "\n";
    this->ui.all_stats->setText(text);
    const Vector<PlayerStatWrap> & players = this->stats->get_players();
    ui.chosenStats_2->insertItem(0, "");
    for (int i = 0; i<players.size(); i++)
    {
        ui.chosenStats_2->insertItem(i+1, players[i].get_name().c_str());
    }
}

void statsGuiMain::clear()
{
    this->stats->clear();
}

void statsGuiMain::player_stats(int index)
{
    index = index--;
    const Vector<PlayerStatWrap> & players = this->stats->get_players();
    const PlayerStatWrap & player = players[index];
    std::string mes="The player: "+ player.get_name()+ "\n wins: "
      +std::to_string(player.get_wins()) + " times \n loose: " +
        std::to_string(player.get_looses()) + " times \n played: "
          +std::to_string(player.get_gamesPlayed()) +
        " games \n for time: "+ std::to_string(player.get_timePlayed()/60) +"
          min";
    this->ui.Info_2->setText(mes.c_str());
    this->ui.Info_2->selectAll();
    this->ui.Info_2->setFontPointSize(12);

}

void statsGuiMain::spin(int maxSel)
{
    PlayerStats::Info<int> info;
    QString text = "name: ";
    QString afer_text = "";
    switch (maxSel) {
    case 0://wins
        info = this->stats->getMaxWins();
```

```
        text = QString(info.owner.c_str()) + " got max wins of: ";
        break;
    case 1://losses
        info = this->stats->getMaxLooses();
        text = QString(info.owner.c_str()) + " got max losses of: ";
        break;
    case 2://games
        info = this->stats->getMaxGamesPlayed();
        text = QString(info.owner.c_str()) + " got max games of: ";
        break;
    case 3://time
        info = this->stats->getMaxTimePlayed();
        info.value /= 60;
        text = QString(info.owner.c_str()) + " got max game time of: ";
        afer_text = QString(" min");
        break;
    }


    text = text + QString(std::to_string(info.value).c_str())+ afer_text;
    this->ui.Info->setText(text);
}

```

Move.h


```
1  #pragma once
2  #include <QWidget>
3  #include "ui_move.h"
4  #include <iostream>
5  #include <QDebug>
6  #include "Country.h"
7  class Move : public QWidget {
8      Q_OBJECT
9
10 public:
11     Move(QWidget * parent = Q_NULLPTR);
12     ~Move();
13
14     void setMax(int);
15     void setMin(int);
16     int getBonusTroops(int sel = 0);
17     void setFrom(Country*);
18     void setTo(Country*);
19 private:
20     QWidget* father;
21     Ui::Move ui;
22     int min, max;
23     Country* to;
24     Country* from;
25 private slots:
26     void plus(int sel = 0);
27     void minus(int sel = 0);
28
29     void end();
30 };
```

Move.cpp


```
1  #include "Move.h"
2
3  Move::Move(QWidget * parent) : QWidget(parent){
4      ui.setupUi(this);
5      this->min = 1;
6      this->max = 1;
7      this->to = nullptr;
8      this->from = nullptr;
9      this->ui.troopsP->setText("1");
10     this->ui.ENTER->show();
11     this->ui.MINUS->show();
12 }
13 Move::~Move()
14 {
15
16 }
17
18 void Move::setMax(int max)
19 {
20     this->max = max;
21 }
22
23 void Move::setMin(int min)
24 {
25     this->min = min;
26     this->ui.troopsP->setText(std::to_string(min).c_str());
27 }
28
29 int Move::getBonusTroops(int sel)
30 {
31     std::string text = this->ui.troopsP->toPlainText().toStdString();
32     return std::stoi(text);
33 }
34
35 void Move::setFrom(Country * from)
36 {
37     this->from = from;
38 }
39
40 void Move::setTo(Country * to)
41 {
42     this->to = to;
43 }
44
45
46 void Move::end() {
47     this->to->incTroops(this->getBonusTroops());
48     this->from->incTroops((-1)*(this->getBonusTroops()));
49     this->close();
50     //this->father->setFocus();
51
52 }
53
54 void Move::plus(int sel)
55 {
56     int number = this->getBonusTroops() + 1;
```

```
    if(number <= this->max)
        this->ui.troopsP->setText(std::to_string(number).c_str());
}

void Move::minus(int sel)
{
    int number = this->getBonusTroops() - 1;
    if (number >= this->min)
        this->ui.troopsP->setText(std::to_string(number).c_str());
}

```