Bachelorthesis

Generated by Doxygen 1.9.1

Chapter 1

Bachelor-Thesis-Project

1.1 Build Instructions

OpenCV 3 with contrib modules needs to be installed via CMake. Add a environment variable "OpenCV_DIR" to the path of the corresponding install root directory. Then build the project itself as usual.

1.2 Camera Calibration

To calibrate Camera use pattern.png on a piece of cardboard. To start Calibration add "-c" Commandline Option. To Calibrate you need at least 15 valid pictures of Chessboard pattern. To save a picture while in calibration mode press Space. After at least 15 valid Pictures (around 50 are recommended) press Enter to start calibration Process.

1.3 Marker Generation

To generate Aruco Markers add "-g" Commandline Option. You can Change quantity of Markers and Markerlibrary in the createArucoMarker() Function (file `main.cpp`).

1.4 Run Program

After you generated Aruco Markers and printed them out you can remove Commandline Options and start the Program. Aruco Markers 1-15 (except 4-6) are implemented atm. Corresponding Munchkincards can be found in `MunchkinCards.cpp`.

There are some bugs left since this is only a Proof of Concept. Sometimes Button have to be pressed multiple Times to work. Random Generator to run away mostly lets you run away.

`https://th-koeln.sciebo.de/s/9oSsSW6dSUsiHBE`

Chapter 2

Namespace Index

2.1 Namespace List

Here is a list of all namespaces with brief descriptions:

Chapter 3

Class Index

3.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

Chapter 4

File Index

4.1 File List

Here is a list of all files with brief descriptions:

Chapter 5

Namespace Documentation

5.1 badthings Namespace Reference

Namespace containing all bad things functions. These can be assigned to a specific card.

Functions

- BadThingsRetVal emptyBadThings (GameState &gamestate, const MunchkinCard &card)
  Does nothing (placeholder)
- BadThingsRetVal looseHand (GameState &gamestate, const MunchkinCard &card)
  Player looses 1 hand and the function checks whether or not he can still carry all his items that need hands.
- BadThingsRetVal looseClass (GameState &gamestate, const MunchkinCard &card)
  Player looses his class.
- BadThingsRetVal playerDies (GameState &gamestate, const MunchkinCard &card)
  Player dies.
- BadThingsRetVal looseLevel (GameState &gamestate, const MunchkinCard &card)
  Player looses 1 level.
- BadThingsRetVal maleDeadFemaleLevelDown (GameState &gamestate, const MunchkinCard &card)
  Male Players die. Female players loose 1 level.

5.1.1 Detailed Description

Namespace containing all bad things functions. These can be assigned to a specific card.

5.1.2 Function Documentation

5.1.2.1 emptyBadThings()

```
BadThingsRetVal badthings::emptyBadThings (
            GameState & gamestate,
            const MunchkinCard & card )
```

Does nothing (placeholder)

Parameters

gamestate	The game state is modified according to the corresponding munchkin rule/card.
card	Used to supply additional information about the card.

5.1.2.2 looseClass()

```
BadThingsRetVal badthings::looseClass (
            GameState & gamestate,
            const MunchkinCard & card )
```

Player looses his class.

Parameters

gamestate	The game state is modified according to the corresponding munchkin rule/card.
card	Used to supply additional information about the card.

5.1.2.3 looseHand()

```
BadThingsRetVal badthings::looseHand (
            GameState & gamestate,
            const MunchkinCard & card )
```

Player looses 1 hand and the function checks whether or not he can still carry all his items that need hands.

Parameters

gamestate	The game state is modified according to the corresponding munchkin rule/card.
card	Used to supply additional information about the card.

5.1.2.4 looseLevel()

```
BadThingsRetVal badthings::looseLevel (
            GameState & gamestate,
            const MunchkinCard & card )
```

Player looses 1 level.

Parameters

gamestate	The game state is modified according to the corresponding munchkin rule/card.
card	Used to supply additional information about the card.

5.1.2.5 maleDeadFemaleLevelDown()

```
BadThingsRetVal badthings::maleDeadFemaleLevelDown (
            GameState & gamestate,
            const MunchkinCard & card )
```

Male Players die. Female players loose 1 level.

Parameters

gamestate	The game state is modified according to the corresponding munchkin rule/card.
card	Used to supply additional information about the card.

5.1.2.6 playerDies()

```
BadThingsRetVal badthings::playerDies (
            GameState & gamestate,
            const MunchkinCard & card )
```

Player dies.

Parameters

gamestate	The game state is modified according to the corresponding munchkin rule/card.
card	Used to supply additional information about the card.

5.2 cardtypeaction Namespace Reference

Namespace containing all cardtype functions. These are mapped to the actual card type.

Functions

- CardTypeRetVal curse (GameState &gamestate, const MunchkinCard &card)
  processes card type curse
- CardTypeRetVal joker (GameState &gamestate, const MunchkinCard &card)
  processes card type joker
- CardTypeRetVal monster (GameState &gamestate, const MunchkinCard &card)
  proceses card type monster
- CardTypeRetVal munchClass (GameState &gamestate, const MunchkinCard &card)
  processes card Type munchkin class
- CardTypeRetVal race (GameState &gamestate, const MunchkinCard &card)

processes card type munchkin race

- CardTypeRetVal item (GameState &gamestate, const MunchkinCard &card)

  processes card type item
- CardTypeRetVal itemBuff (GameState &gamestate, const MunchkinCard &card)

  processes card type item buff
- CardTypeRetVal lvlUp (GameState &gamestate, const MunchkinCard &card)

  processes card type level up

5.2.1 Detailed Description

Namespace containing all cardtype functions. These are mapped to the actual card type.

5.2.2 Function Documentation

5.2.2.1 curse()

```
CardTypeRetVal cardtypeaction::curse (
            GameState & gamestate,
            const MunchkinCard & card )
```

processes card type curse

Parameters

gamestate	The game state is modified according to the corresponding munchkin rule/card.
card	Used to supply additional information about the card.

5.2.2.2 item()

```
CardTypeRetVal cardtypeaction::item (
            GameState & gamestate,
            const MunchkinCard & card )
```

processes card type item

Parameters

gamestate	The game state is modified according to the corresponding munchkin rule/card.
card	Used to supply additional information about the card.

5.2.2.3 itemBuff()

```
CardTypeRetVal cardtypeaction::itemBuff (
            GameState & gamestate,
            const MunchkinCard & card )
```

processes card type item buff

Parameters

gamestate	The game state is modified according to the corresponding munchkin rule/card.
card	Used to supply additional information about the card.

5.2.2.4 joker()

```
CardTypeRetVal cardtypeaction::joker (
            GameState & gamestate,
            const MunchkinCard & card )
```

processes card type joker

Parameters

gamestate	The game state is modified according to the corresponding munchkin rule/card.
card	Used to supply additional information about the card.

5.2.2.5 lvlUp()

```
CardTypeRetVal cardtypeaction::lvlUp (
            GameState & gamestate,
            const MunchkinCard & card )
```

processes card type level up

Parameters

gamestate	The game state is modified according to the corresponding munchkin rule/card.
card	Used to supply additional information about the card.

5.2.2.6 monster()

```
CardTypeRetVal cardtypeaction::monster (
```

```
            GameState & gamestate,
            const MunchkinCard & card )
```

proceses card type monster

Parameters

gamestate	The game state is modified according to the corresponding munchkin rule/card.
card	Used to supply additional information about the card.

5.2.2.7 munchClass()

```
CardTypeRetVal cardtypeaction::munchClass (
            GameState & gamestate,
            const MunchkinCard & card )
```

processes card Type munchkin class

Parameters

gamestate	The game state is modified according to the corresponding munchkin rule/card.
card	Used to supply additional information about the card.

5.2.2.8 race()

```
CardTypeRetVal cardtypeaction::race (
            GameState & gamestate,
            const MunchkinCard & card )
```

processes card type munchkin race

Parameters

gamestate	The game state is modified according to the corresponding munchkin rule/card.
card	Used to supply additional information about the card.

5.3 constants Namespace Reference

Variables

- constexpr int yolo_grid_size = 13

5.3.1 Variable Documentation

5.3.1.1 yolo_grid_size

```
constexpr int constants::yolo_grid_size = 13  [inline], [constexpr]
```

5.4 extras Namespace Reference

5.5 munch_tut Namespace Reference

namespace for munchkin tutorial

Typedefs

- using Color = std::array< int, 3 >

5.5.1 Detailed Description

namespace for munchkin tutorial

5.5.2 Typedef Documentation

5.5.2.1 Color

```
using munch_tut::Color = typedef std::array<int, 3>
```

Chapter 6

Class Documentation

6.1 BadThingsRetVal Struct Reference

`#include <BadThings.h>`

The documentation for this struct was generated from the following file:

- BadThings.h

6.2 Button Class Reference

class for Buttons

`#include <Button.h>`

Public Member Functions

- Button (int _id, const cv::Rect &_rect, const cv::Scalar &_color, bool _visible=true, ButtonOrigin _origin=ButtonOrigin::topleft, const button_callback &_callback={})

  constructor for a button
- Button (const Button &)=default

  Default copy constructor.
- Button (Button &&)=default

  Default move constructor.
- Button & operator= (const Button &)=default

  Default copy assignment.
- Button & operator= (Button &&)=default

  Default move assignment.
- bool poll_click (const cv::Point &point, const cv::Size &canvas_size) const

  function to check if click was inside the button
- void draw (cv::Mat &canvas) const

  draw function for buttons

Public Attributes

- int id

  Button id.
- cv::Rect rect

  rectangle that defines size of button
- button_callback callback

  callback function for button to apply action to button
- cv::Scalar color

  color of the button
- bool visible

  wether or not button should be visible
- ButtonOrigin origin

  position of the button

6.2.1 Detailed Description

class for Buttons

6.2.2 Constructor & Destructor Documentation

6.2.2.1 Button() [1/3]

```
Button::Button (
            int _id,
            const cv::Rect & _rect,
            const cv::Scalar & _color,
            bool _visible = true,
            ButtonOrigin _origin = ButtonOrigin::topleft,
            const button_callback & _callback = {} )
```

constructor for a button

Parameters

_id	Button id
_rect	rectangle which defines the dimensions of the button
_color	defines the color of the button in GBR
_visible	used to hide button when not needed
_origin	origin of the button as in enum class ButtonOrigin
_callback	callback function for button. Can be changed so that one button can serve mulitple purposes if necessary

6.2.2.2 Button() [2/3]

```
Button::Button (
            const Button &  )  [default]
```

Default copy constructor.

6.2.2.3 Button() [3/3]

```
Button::Button (
            Button &&  )  [default]
```

Default move constructor.

6.2.3 Member Function Documentation

6.2.3.1 draw()

```
void Button::draw (
            cv::Mat & canvas ) const
```

draw function for buttons

Parameters

canvas	defines the button that should be drawn

6.2.3.2 operator=() [1/2]

```
Button& Button::operator= (
            Button &&  )  [default]
```

Default move assignment.

6.2.3.3 operator=() [2/2]

```
Button& Button::operator= (
            const Button &  )  [default]
```

Default copy assignment.

6.2.3.4 poll_click()

```
bool Button::poll_click (
            const cv::Point & point,
            const cv::Size & canvas_size ) const
```

function to check if click was inside the button

Parameters

point	point of click
canvas_size	size of the canvas that defines the button size

6.2.4 Member Data Documentation

6.2.4.1 callback

```
button_callback Button::callback
```

callback function for button to apply action to button

6.2.4.2 color

```
cv::Scalar Button::color
```

color of the button

6.2.4.3 id

```
int Button::id
```

Button id.

6.2.4.4 origin

```
ButtonOrigin Button::origin
```

position of the button

6.2.4.5 rect

```
cv::Rect Button::rect
```

rectangle that defines size of button

6.2.4.6 visible

```
bool Button::visible
```

wether or not button should be visible

The documentation for this class was generated from the following files:

- Button.h
- Button.cpp

6.3 CardTypeRetVal Struct Reference

```
#include <CardTypeActions.h>
```

The documentation for this struct was generated from the following file:

- CardTypeActions.h

6.4 ExtrasRetVal Struct Reference

```
#include <Extras.h>
```

The documentation for this struct was generated from the following file:

- Extras.h

6.5 GameState Struct Reference

```
#include <GameState.h>
```

Public Attributes

- PlayerStats player01

  playerStats object for data of player stats that are needed
- MouseParams mouseparams

  mouseParams object for data of tutorial that are needed e.g. tutorial text
- std::vector< NNresult > nnResults

  vector of NNresult objects to store all card found by YOLO Neuronal Net
- std::vector< Button > buttons

  buttons that are needed for the tutorial
- bool should_exit

  param wether or not the game should exit after exit button was pressed
- bool should_continue

  param for when user input is awaited and the tutorial needs to jump to a different point in logic function
- bool end_turn

  param to signal the player that the end of the turn is reached
- bool run_away

  param for when the user needs to run away and the tutorial triggers an random number as dice roll
- bool remove_card

  param for when the player chooses or needs to get rid of a card he has equiped
- cv::Size canvas_size

  param of the canvas size used for the buttons

6.5.1 Detailed Description

data of all needed params for the tutorial

6.5.2 Member Data Documentation

6.5.2.1 buttons

```
std::vector<Button> GameState::buttons
```

buttons that are needed for the tutorial

6.5.2.2 canvas_size

```
cv::Size GameState::canvas_size
```

param of the canvas size used for the buttons

6.5.2.3 end_turn

```
bool GameState::end_turn
```

param to signal the player that the end of the turn is reached

6.5.2.4 mouseparams

```
MouseParams GameState::mouseparams
```

mouseParams object for data of tutorial that are needed e.g. tutorial text

6.5.2.5 nnResults

```
std::vector<NNresult> GameState::nnResults
```

vector of NNresult objects to store all card found by YOLO Neuronal Net

6.5.2.6 player01

```
PlayerStats GameState::player01
```

playerStats object for data of player stats that are needed

6.5.2.7 remove_card

```
bool GameState::remove_card
```

param for when the player chooses or needs to get rid of a card he has equiped

6.5.2.8 run_away

```
bool GameState::run_away
```

param for when the user needs to run away and the tutorial triggers an random number as dice roll

6.5.2.9 should_continue

```
bool GameState::should_continue
```

param for when user input is awaited and the tutorial needs to jump to a different point in logic function

6.5.2.10 should_exit

```
bool GameState::should_exit
```

param wether or not the game should exit after exit button was pressed

The documentation for this struct was generated from the following file:

- GameState.h

6.6 InputEvent Struct Reference

```
#include <InputEvent.h>
```

Public Attributes

- EventType type

6.6.1 Member Data Documentation

6.6.1.1 type

```
EventType InputEvent::type
```

The documentation for this struct was generated from the following file:

- InputEvent.h

6.7 MouseParams Struct Reference

Holds data of a mouse event.

```
#include <GameState.h>
```

Public Attributes

- vector< cv::Point > poly
  Polygon of the clicked marker.
- int markerId
  Id of the marker.
- vector< cv::Point2f > markerCorner
  Four corners of the marker.
- vector< string > tutText
  Tutorial text to display for the marker.
- cv::Point clickP
  Mouse position.
- MouseEvent event
  Mouse event type.

6.7.1 Detailed Description

Holds data of a mouse event.

6.7.2 Member Data Documentation

6.7.2.1 clickP

```
cv::Point MouseParams::clickP
```

Mouse position.

6.7.2.2 event

```
MouseEvent MouseParams::event
```

Mouse event type.

6.7.2.3 markerCorner

```
vector<cv::Point2f> MouseParams::markerCorner
```

Four corners of the marker.

6.7.2.4 markerId

```
int MouseParams::markerId
```

Id of the marker.

6.7.2.5 poly

```
vector<cv::Point> MouseParams::poly
```

Polygon of the clicked marker.

6.7.2.6 tutText

```
vector<string> MouseParams::tutText
```

Tutorial text to display for the marker.

The documentation for this struct was generated from the following file:

- GameState.h

6.8 MunchkinCard Class Reference

class of munchkin cards with params that could be needed for a single card

```
#include <MunchkinCards.h>
```

Public Member Functions

- MunchkinCard ()
  Default constructor.
- MunchkinCard (int _markerID, const string &_cardName, const string &_effect, const string &_badThings, const string &_itemEffect, const string &_itemNeeds, vector< string > _bonis, ParentCardType _parentCardType, CardType _type, ItemType _itemType, int _strengthBoni, int _debuff, int _monsStrength, int _lvlUp, int _treasures, int _itemValue, int _handsNeeded, bool _itemLarge)
  custom constructor
- MunchkinCard (const MunchkinCard &other)=default
  default copy constructor
- MunchkinCard (MunchkinCard &&other)=default
  default move constructor
- MunchkinCard & operator= (const MunchkinCard &other)=default
  default copy assignment
- MunchkinCard & operator= (MunchkinCard &&other)=default
  default move assignment

Static Public Member Functions

- static vector< MunchkinCard > cardsConstr ()

  function to create all cards from MunchkinCards.cpp

Public Attributes

- string cardName

  card name
- string effect

  card effect
- vector< string > bonis

  boni that the card can have
- string badThings

  (deprecated) string of bad things
- string itemEffect

  effect that an item has e.g. attack with fire and flame
- string itemNeeds

  param that a munchkin needs to equip a item e.g. has to be a dwarf
- ParentCardType parentCardType

  param for parent card type
- CardType type

  param for "child" card type
- ItemType itemType

  item type
- int markerID

  marker id that corresponse with the card
- int strengthBoni

  how much strength the munchkin receives with this card
- int debuff

  how much strength the munchkin looses through this card
- int monsStrength

  param how much the monsters strength is
- int lvlUp

  how many level you get r.g. when defeating the monster
- int treasures

  how many treasure the munchkin gets when defeating the monster
- int itemValue

  how much gold the item sells for
- int handsNeeded

  how many hands a munchkin needs to equip the item
- bool itemLarge

  wether or not the item is large
- BadThingsFunc badThingsFunc

  bad things function that mapps onto a bad things function in badThingsFunc.h

6.8.1 Detailed Description

class of munchkin cards with params that could be needed for a single card

6.8.2 Constructor & Destructor Documentation

6.8.2.1 MunchkinCard() [1/4]

```
MunchkinCard::MunchkinCard ( )  [inline]
```

Default constructor.

6.8.2.2 MunchkinCard() [2/4]

```
MunchkinCard::MunchkinCard (
            int _markerID,
            const string & _cardName,
            const string & _effect,
            const string & _badThings,
            const string & _itemEffect,
            const string & _itemNeeds,
            vector< string > _bonis,
            ParentCardType _parentCardType,
            CardType _type,
            ItemType _itemType,
            int _strengthBoni,
            int _debuff,
            int _monsStrength,
            int _lvlUp,
            int _treasures,
            int _itemValue,
            int _handsNeeded,
            bool _itemLarge )
```

custom constructor

Parameters

_markerID	marker id that corresponse with the card
_cardName	card name
_effect	card effect
_badThings	(deprecated) string of bad things
_itemEffect	effect that an item has e.g. attack with fire and flame
_itemNeeds	param that a munchkin needs to equip a item e.g. has to be a dwarf
_bonis	boni that the card can have
_parentCardType	param for parent card type
_type	param for "child" card type
_itemType	item type
_strengthBoni	how much strength the munchkin receives with this card
_debuff	how much strength the munchkin looses through this card
_monsStrength	param how much the monsters strength is

Parameters

_lvlUp	how many level you get r.g. when defeating the monster
_treasures	how many treasure the munchkin gets when defeating the monster
_itemValue	how much gold the item sells for
_handsNeeded	how many hands a munchkin needs to equip the item
_itemLarge	wether or not the item is large

6.8.2.3 MunchkinCard() [3/4]

```
MunchkinCard::MunchkinCard (
            const MunchkinCard & other )  [default]
```

default copy constructor

6.8.2.4 MunchkinCard() [4/4]

```
MunchkinCard::MunchkinCard (
            MunchkinCard && other )  [default]
```

default move constructor

6.8.3 Member Function Documentation

6.8.3.1 cardsConstr()

```
vector< MunchkinCard > MunchkinCard::cardsConstr ( )  [static]
```

function to create all cards from MunchkinCards.cpp

6.8.3.2 operator=() [1/2]

```
MunchkinCard& MunchkinCard::operator= (
            const MunchkinCard & other )  [default]
```

default copy assignment

6.8.3.3 operator=() [2/2]

```
MunchkinCard& MunchkinCard::operator= (
            MunchkinCard && other )  [default]
```

default move assignment

6.8.4 Member Data Documentation

6.8.4.1 badThings

```
string MunchkinCard::badThings
```

(deprecated) string of bad things

6.8.4.2 badThingsFunc

```
BadThingsFunc MunchkinCard::badThingsFunc
```

bad things function that mapps onto a bad things function in badThingsFunc.h

6.8.4.3 bonis

```
vector<string> MunchkinCard::bonis
```

boni that the card can have

6.8.4.4 cardName

```
string MunchkinCard::cardName
```

card name

6.8.4.5 debuff

```
int MunchkinCard::debuff
```

how much strength the munchkin looses through this card

6.8.4.6 effect

```
string MunchkinCard::effect
```

card effect

6.8.4.7 handsNeeded

```
int MunchkinCard::handsNeeded
```

how many hands a munchkin needs to equip the item

6.8.4.8 itemEffect

```
string MunchkinCard::itemEffect
```

effect that an item has e.g. attack with fire and flame

6.8.4.9 itemLarge

```
bool MunchkinCard::itemLarge
```

wether or not the item is large

6.8.4.10 itemNeeds

```
string MunchkinCard::itemNeeds
```

param that a munchkin needs to equip a item e.g. has to be a dwarf

6.8.4.11 itemType

```
ItemType MunchkinCard::itemType
```

item type

6.8.4.12 itemValue

```
int MunchkinCard::itemValue
```

how much gold the item sells for

6.8.4.13 lvlUp

```
int MunchkinCard::lvlUp
```

how many level you get r.g. when defeating the monster

6.8.4.14 markerID

```
int MunchkinCard::markerID
```

marker id that corresponse with the card

6.8.4.15 monsStrength

```
int MunchkinCard::monsStrength
```

param how much the monsters strength is

6.8.4.16 parentCardType

```
ParentCardType MunchkinCard::parentCardType
```

param for parent card type

6.8.4.17 strengthBoni

```
int MunchkinCard::strengthBoni
```

how much strength the munchkin receives with this card

6.8.4.18 treasures

```
int MunchkinCard::treasures
```

how many treasure the munchkin gets when defeating the monster

6.8.4.19 type

```
CardType MunchkinCard::type
```

param for "child" card type

The documentation for this class was generated from the following files:

- MunchkinCards.h
- MunchkinCards.cpp

6.9 NNresult Struct Reference

struct that holds custom result from YOLO Neuronal Net

```
#include <GameState.h>
```

Public Attributes

- int id
  id of NN result class which translates to recognized card
- cv::Rect nnBBox
  bounding box of recognized NN result which highlights the card

6.9.1 Detailed Description

struct that holds custom result from YOLO Neuronal Net

6.9.2 Member Data Documentation

6.9.2.1 id

```
int NNresult::id
```

id of NN result class which translates to recognized card

6.9.2.2 nnBBox

```
cv::Rect NNresult::nnBBox
```

bounding box of recognized NN result which highlights the card

The documentation for this struct was generated from the following file:

- GameState.h

6.10 PlayerStats Struct Reference

Holds data of player stats.

```
#include <GameState.h>
```

Public Attributes

- string sex
  sex of player (male/female)
- vector< string > bonis
  bonis for player that are gained through different cards e.g. munchkin class
- vector< string > curses
  permanent curses that are a burden for the player
- vector< string > itemEffects
  effects of different items that are supporting the player e.g. attack with fire
- vector< string > munchClasses
  munchkin classes the player optained
- vector< string > munchRaces
  munchkin races the player optained
- int lvl
  level of the player
- int strength
  strength of a player through his equipment
- int hands
  how many hands player has left that are not holding an object
- int runStrength
  dice roll hat to be larger than x for the player to run away
- int availableClasses
  how many classes can the player optain
- int availableRaces
  how many races can the player optain
- bool carriesLargeItems
  wether or not the player carries a large item
- bool hasArmor
  wether or not the player carries an armor
- bool hasHat
  wether or not the player carries a hat
- bool hasShoes
  wether or not the player carries shoes

6.10.1 Detailed Description

Holds data of player stats.

6.10.2 Member Data Documentation

6.10.2.1 availableClasses

```
int PlayerStats::availableClasses
```

how many classes can the player optain

6.10.2.2 availableRaces

```
int PlayerStats::availableRaces
```

how many races can the player optain

6.10.2.3 bonis

```
vector<string> PlayerStats::bonis
```

bonis for player that are gained through different cards e.g. munchkin class

6.10.2.4 carriesLargeItems

```
bool PlayerStats::carriesLargeItems
```

wether or not the player carries a large item

6.10.2.5 curses

```
vector<string> PlayerStats::curses
```

permanent curses that are a burden for the player

6.10.2.6 hands

```
int PlayerStats::hands
```

how many hands player has left that are not holding an object

6.10.2.7 hasArmor

```
bool PlayerStats::hasArmor
```

wether or not the player carries an armor

6.10.2.8 hasHat

```
bool PlayerStats::hasHat
```

wether or not the player carries a hat

6.10.2.9 hasShoes

```
bool PlayerStats::hasShoes
```

wether or not the player carries shoes

6.10.2.10 itemEffects

```
vector<string> PlayerStats::itemEffects
```

effects of different items that are supporting the player e.g. attack with fire

6.10.2.11 lvl

```
int PlayerStats::lvl
```

level of the player

6.10.2.12 munchClasses

```
vector<string> PlayerStats::munchClasses
```

munchkin classes the player optained

6.10.2.13 munchRaces

```
vector<string> PlayerStats::munchRaces
```

munchkin races the player optained

6.10.2.14 runStrength

```
int PlayerStats::runStrength
```

dice roll hat to be larger than x for the player to run away

6.10.2.15 sex

```
string PlayerStats::sex
```

sex of player (male/female)

6.10.2.16 strength

```
int PlayerStats::strength
```

strength of a player through his equipment

The documentation for this struct was generated from the following file:

- GameState.h

6.11 testOutput Struct Reference

struct for object that is used for the YOLO tests.

```
#include <common.h>
```

Public Attributes

- vector< stringstream > singleFile

  vector of stringstreams for single test output files
- vector< stringstream > csvSingleFile

  vector of stringstreams for single test output csv files
- stringstream fileString

  stringstream for the whole test output file
- stringstream csvFileString

  stringstream for the whole test output csv file
- int fileCounter

  counter for how many files are already written
- int debugTick

  current tick (seconds) for test
- int debugSingleOutput

  tick value for when a single output file should be generated
- double allClasses

  adds up the munchkin classes found in a single frame to calculate the average class count
- double inferenceTime

  adds up the inference time for each frame to calculate the average inferance time
- double allPropability

  double value to add up all class propabilities to calculate the average class propability
- string YoloName

  name of yolo net that is tested
- string MunchkinClassesNum

  Number of munchkin classes (cards) that are used in the test.

6.11.1 Detailed Description

struct for object that is used for the YOLO tests.

6.11.2 Member Data Documentation

6.11.2.1 allClasses

```
double testOutput::allClasses
```

adds up the munchkin classes found in a single frame to calculate the average class count

6.11.2.2 allPropability

```
double testOutput::allPropability
```

double value to add up all class propabilities to calculate the average class propability

6.11.2.3 csvFileString

```
stringstream testOutput::csvFileString
```

stringstream for the whole test output csv file

6.11.2.4 csvSingleFile

```
vector<stringstream> testOutput::csvSingleFile
```

vector of stringstreams for single test output csv files

6.11.2.5 debugSingleOutput

```
int testOutput::debugSingleOutput
```

tick value for when a single output file should be generated

6.11.2.6 debugTick

```
int testOutput::debugTick
```

current tick (seconds) for test

6.11.2.7 fileCounter

```
int testOutput::fileCounter
```

counter for how many files are already written

6.11.2.8 fileString

```
stringstream testOutput::fileString
```

stringstream for the whole test output file

6.11.2.9 inferenceTime

```
double testOutput::inferenceTime
```

adds up the inference time for each frame to calculate the average inferance time

6.11.2.10 MunchkinClassesNum

```
string testOutput::MunchkinClassesNum
```

Number of munchkin classes (cards) that are used in the test.

6.11.2.11 singleFile

```
vector<stringstream> testOutput::singleFile
```

vector of stringstreams for single test output files

6.11.2.12 YoloName

```
string testOutput::YoloName
```

name of yolo net that is tested

The documentation for this struct was generated from the following file:

- common.h

Chapter 7

File Documentation

7.1 ArucoGenerator.cpp File Reference

```
#include <ArucoGenerator.h>
```

Functions

- void createArucoMarkers ()

  function to generate Aruco Markers. Dictionary for Aruco Markers are choosen inside this function.

7.1.1 Function Documentation

7.1.1.1 createArucoMarkers()

```
void createArucoMarkers ( )
```

function to generate Aruco Markers. Dictionary for Aruco Markers are choosen inside this function.

7.2 ArucoGenerator.h File Reference

class to generate new Aruco Markers

```
#include <vector>
#include "opencv2/opencv.hpp"
#include "opencv2/aruco.hpp"
#include <string>
#include <sstream>
#include <iostream>
#include <fstream>
```

Functions

- void createArucoMarkers ()

  function to generate Aruco Markers. Dictionary for Aruco Markers are choosen inside this function.

7.2.1 Detailed Description

class to generate new Aruco Markers

Author

Benjamin Lueben

In this class the Aruco Markers can be generated. By now 50 4x4 Aruco Markers are generated.

7.2.2 Function Documentation

7.2.2.1 createArucoMarkers()

```
void createArucoMarkers ( )
```

function to generate Aruco Markers. Dictionary for Aruco Markers are choosen inside this function.

7.3 BadThings.cpp File Reference

```
#include <BadThings.h>
#include <MunchkinCards.h>
#include <GameState.h>
```

7.4 BadThings.h File Reference

Bad Things Functions for Munchkin Cards.

```
#include <functional>
```

Classes

- struct BadThingsRetVal

Namespaces

- badthings

  Namespace containing all bad things functions. These can be assigned to a specific card.

Typedefs

- using BadThingsFunc = std::function< BadThingsRetVal(GameState &, const MunchkinCard &)>

  Callback type for bad things behaviour.

Functions

- BadThingsRetVal badthings::emptyBadThings (GameState &gamestate, const MunchkinCard &card)

  Does nothing (placeholder)
- BadThingsRetVal badthings::looseHand (GameState &gamestate, const MunchkinCard &card)

  Player looses 1 hand and the function checks whether or not he can still carry all his items that need hands.
- BadThingsRetVal badthings::looseClass (GameState &gamestate, const MunchkinCard &card)

  Player looses his class.
- BadThingsRetVal badthings::playerDies (GameState &gamestate, const MunchkinCard &card)

  Player dies.
- BadThingsRetVal badthings::looseLevel (GameState &gamestate, const MunchkinCard &card)

  Player looses 1 level.
- BadThingsRetVal badthings::maleDeadFemaleLevelDown (GameState &gamestate, const MunchkinCard &card)

  Male Players die. Female players loose 1 level.

7.4.1 Detailed Description

Bad Things Functions for Munchkin Cards.

Author

Benjamin Lueben

In this class all the BadThings functions for every Munchkin Card are defined.

7.4.2 Typedef Documentation

7.4.2.1 BadThingsFunc

```
using BadThingsFunc = std::function<BadThingsRetVal(GameState&, const MunchkinCard&)>
```

Callback type for bad things behaviour.

7.5 Button.cpp File Reference

```
#include "Button.h"
```

7.6 Button.h File Reference

Class to define buttons which are used for user input.

```
#include "opencv2/opencv.hpp"
#include "opencv2/imgcodecs.hpp"
#include "opencv2/imgproc.hpp"
#include <functional>
```

Classes

- class Button
  class for Buttons

Typedefs

- using button_callback = std::function< void(const Button &)>

Enumerations

- enum class ButtonOrigin { topleft , topright , bottomleft , bottomright }
  specifies in what corner the button should be shown

7.6.1 Detailed Description

Class to define buttons which are used for user input.

Author

Benjamin Lueben

With this class different buttons can be defined which can be used for user input. Buttons are used when the user needs to confirm an action or to exit the tutorial.

7.6.2 Typedef Documentation

7.6.2.1 button_callback

```
using button_callback = std::function<void(const Button&)>
```

7.6.3 Enumeration Type Documentation

7.6.3.1 ButtonOrigin

```
enum ButtonOrigin  [strong]
```

specifies in what corner the button should be shown

Enumerator

topleft	top left corner
topright	top right corner
bottomleft	bottom left corner
bottomright	bottom right corner

7.7 CameraCalibration.cpp File Reference

```
#include <CameraCalibration.h>
```

Functions

- void createKnownBoardPositions (Size boardSize, float squareEdgeLength, vector< Point3f > &corners)

  creates known boards position for chessboard used to calibrate camera.
- void getChessboardCorners (vector< Mat > images, vector< vector< Point2f >> &allFoundCorners, bool showResults)

  function to get chessboard corners from given images.
- bool saveCameraCalibration (string name, Mat cameraMatrix, Mat distanceCoefficients)

  function to save camera calibration to calibration file.
- bool loadCameraCalibration (string name, Mat &cameraMatrix, Mat &distanceCoefficients)

  function so load camera calibration from file
- void cameraCalibration (vector< Mat > calibrationImages, Size boardSize, float squareEdgeLength, Mat &cameraMatrix, Mat &distanceCoefficients)

  camera calibration function to process the actual camera calibration.
- void cameraCalibrationProcess (Mat &cameraMatrix, Mat &distanceCoefficients, Size chessboardDimensions, float calbirationSquareDimension)

  function to guide through the camera calibration process. Can take Pictures of chess board to calibrate camera with

7.7.1 Function Documentation

7.7.1.1 cameraCalibration()

```
void cameraCalibration (
            vector< Mat > calibrationImages,
            Size boardSize,
            float squareEdgeLength,
            Mat & cameraMatrix,
            Mat & distanceCoefficients )
```

camera calibration function to process the actual camera calibration.

Parameters

calibrationImages	vector of mat objects that store the calibration images.
boardSize	size of chess board used to calibrate camera e.g 9x6
squareEdgeLength	edge length of single square tile on calibration board
cameraMatrix	reference of mat object for camera matrix to be stored in
distanceCoefficients	reference of mat object for distance coefficients to be stored in

7.7.1.2 cameraCalibrationProcess()

```
void cameraCalibrationProcess (
            Mat & cameraMatrix,
            Mat & distanceCoefficients,
            Size chessboardDimensions,
            float calbirationSquareDimension )
```

function to guide through the camera calibration process. Can take Pictures of chess board to calibrate camera with

Parameters

cameraMatrix	reference of mat object to store camera matrix in after calibration.
distanceCoefficients	reference of mat object to store distance coefficients in after calibration.
chessboardDimensions	size of chessboard dimensions e.g 9x6
calibrationSquareDimensions	edge length of single square tile on calibration board

7.7.1.3 createKnownBoardPositions()

```
void createKnownBoardPositions (
            Size boardSize,
```

```
            float squareEdgeLength,
            vector< Point3f > & corners )
```

creates known boards position for chessboard used to calibrate camera.

Parameters

boardSize	size of the calibration chess board. e.g 9x6
squareEdgeLength	Square edge length of single calibration board tile. Needs to be meassured from printed chess board.
corners	reference of chessboard corners that are returned.

7.7.1.4 getChessboardCorners()

```
void getChessboardCorners (
            vector< Mat > images,
            vector< vector< Point2f >> & allFoundCorners,
            bool showResult = false )
```

function to get chessboard corners from given images.

Parameters

images	Vector of Mat objects with calibration pictures of chessboard.
allFoundCorners	vector with vector of Points that return all found corners of tiles inside the chessboard pictures.
showResult	bool of wether or not the results should be shown on the output image.

7.7.1.5 loadCameraCalibration()

```
bool loadCameraCalibration (
            string name,
            Mat & cameraMatrix,
            Mat & distanceCoefficients )
```

function so load camera calibration from file

Parameters

name	filepath for camera calibration file
cameraMatrix	reference to mat object to store camera matrix in and return camera matrix.
distanceCoefficients	reference to mat object to store distance coefficients in and return them.

Returns

true => file could be opened and camera matrix was loaded; false => file could not be opened.

7.7.1.6 saveCameraCalibration()

```
bool saveCameraCalibration (
            string name,
            Mat cameraMatrix,
            Mat distanceCoefficients )
```

function to save camera calibration to calibration file.

Parameters

name	name for the filepath to store camera matrix in
cameraMatrix	calculated camera matrix mat object to store inside calibration file
distanceCoefficients	distance coefficients that are also stored within the camera calibration file

Returns

true => file was saved successfully; false => file could not be created;

7.8 CameraCalibration.h File Reference

Camera Calibration with OpenCV Chessboard.

```
#include "opencv2/opencv.hpp"
#include "opencv2/imgcodecs.hpp"
#include "opencv2/imgproc.hpp"
#include "opencv2/highgui.hpp"
#include "opencv2/aruco.hpp"
#include "opencv2/calib3d.hpp"
#include "opencv2/video.hpp"
#include "opencv2/videoio.hpp"
```

Functions

- void createKnownBoardPositions (Size boardSize, float squareEdgeLength, vector< Point3f > &corners)

  creates known boards position for chessboard used to calibrate camera.
- void getChessboardCorners (vector< Mat > images, vector< vector< Point2f >> &allFoundCorners, bool showResult=false)

  function to get chessboard corners from given images.
- bool saveCameraCalibration (string name, Mat cameraMatrix, Mat distanceCoefficients)

  function to save camera calibration to calibration file.
- bool loadCameraCalibration (string name, Mat &cameraMatrix, Mat &distanceCoefficients)

  function so load camera calibration from file
- void cameraCalibration (vector< Mat > calibrationImages, Size boardSize, float squareEdgeLength, Mat &cameraMatrix, Mat &distanceCoefficients)

  camera calibration function to process the actual camera calibration.
- void cameraCalibrationProcess (Mat &cameraMatrix, Mat &distanceCoefficients, Size chessboardDimensions, float calbirationSquareDimension)

  function to guide through the camera calibration process. Can take Pictures of chess board to calibrate camera with

7.8.1 Detailed Description

Camera Calibration with OpenCV Chessboard.

Author

Benjamin Lueben

In this class the Webcam gets calibrated for use of Aruco Markers. Calibration is done with OpenCV and Chessboard pattern. This Class is based on a series of youtube tutorials from George Lecakes regarding OpenCV Camera Calibration and Aruco Markers (pt1 of Camera Calibration: https://www.youtube.com/watch?v=HNfPbw-1e_w)

7.8.2 Function Documentation

7.8.2.1 cameraCalibration()

```
void cameraCalibration (
            vector< Mat > calibrationImages,
            Size boardSize,
            float squareEdgeLength,
            Mat & cameraMatrix,
            Mat & distanceCoefficients )
```

camera calibration function to process the actual camera calibration.

Parameters

calibrationImages	vector of mat objects that store the calibration images.
boardSize	size of chess board used to calibrate camera e.g 9x6
squareEdgeLength	edge length of single square tile on calibration board
cameraMatrix	reference of mat object for camera matrix to be stored in
distanceCoefficients	reference of mat object for distance coefficients to be stored in

7.8.2.2 cameraCalibrationProcess()

```
void cameraCalibrationProcess (
            Mat & cameraMatrix,
            Mat & distanceCoefficients,
            Size chessboardDimensions,
            float calbirationSquareDimension )
```

function to guide through the camera calibration process. Can take Pictures of chess board to calibrate camera with

Parameters

cameraMatrix	reference of mat object to store camera matrix in after calibration.
distanceCoefficients	reference of mat object to store distance coefficients in after calibration.
chessboardDimensions	size of chessboard dimensions e.g 9x6
calibrationSquareDimensions	edge length of single square tile on calibration board

7.8.2.3 createKnownBoardPositions()

```
void createKnownBoardPositions (
            Size boardSize,
            float squareEdgeLength,
            vector< Point3f > & corners )
```

creates known boards position for chessboard used to calibrate camera.

Parameters

boardSize	size of the calibration chess board. e.g 9x6
squareEdgeLength	Square edge length of single calibration board tile. Needs to be meassured from printed chess board.
corners	reference of chessboard corners that are returned.

7.8.2.4 getChessboardCorners()

```
void getChessboardCorners (
            vector< Mat > images,
            vector< vector< Point2f >> & allFoundCorners,
            bool showResult = false )
```

function to get chessboard corners from given images.

Parameters

images	Vector of Mat objects with calibration pictures of chessboard.
allFoundCorners	vector with vector of Points that return all found corners of tiles inside the chessboard pictures.
showResult	bool of wether or not the results should be shown on the output image.

7.8.2.5 loadCameraCalibration()

```
bool loadCameraCalibration (
            string name,
```

```
            Mat & cameraMatrix,
            Mat & distanceCoefficients )
```

function so load camera calibration from file

Parameters

name	filepath for camera calibration file
cameraMatrix	reference to mat object to store camera matrix in and return camera matrix.
distanceCoefficients	reference to mat object to store distance coefficients in and return them.

Returns

true => file could be opened and camera matrix was loaded; false => file could not be opened.

7.8.2.6 saveCameraCalibration()

```
bool saveCameraCalibration (
            string name,
            Mat cameraMatrix,
            Mat distanceCoefficients )
```

function to save camera calibration to calibration file.

Parameters

name	name for the filepath to store camera matrix in
cameraMatrix	calculated camera matrix mat object to store inside calibration file
distanceCoefficients	distance coefficients that are also stored within the camera calibration file

Returns

true => file was saved successfully; false => file could not be created;

7.9 CardTypeActions.cpp File Reference

```
#include "CardTypeActions.h"
#include <iostream>
#include "GameState.h"
#include "MunchkinCards.h"
```

Functions

- bool runAway (GameState &gamestate)

7.9.1 Function Documentation

7.9.1.1 runAway()

```
bool runAway (
            GameState & gamestate )
```

7.10 CardTypeActions.h File Reference

Class to define the functions for munchkin card types.

```
#include <functional>
```

Classes

- struct CardTypeRetVal

Namespaces

- cardtypeaction

  Namespace containing all cardtype functions. These are mapped to the actual card type.

Typedefs

- using CardTypeFunc = std::function< CardTypeRetVal(GameState &, const MunchkinCard &)>

Functions

- CardTypeRetVal cardtypeaction::curse (GameState &gamestate, const MunchkinCard &card)

  processes card type curse
- CardTypeRetVal cardtypeaction::joker (GameState &gamestate, const MunchkinCard &card)

  processes card type joker
- CardTypeRetVal cardtypeaction::monster (GameState &gamestate, const MunchkinCard &card)

  proceses card type monster
- CardTypeRetVal cardtypeaction::munchClass (GameState &gamestate, const MunchkinCard &card)

  processes card Type munchkin class
- CardTypeRetVal cardtypeaction::race (GameState &gamestate, const MunchkinCard &card)

  processes card type munchkin race
- CardTypeRetVal cardtypeaction::item (GameState &gamestate, const MunchkinCard &card)

  processes card type item
- CardTypeRetVal cardtypeaction::itemBuff (GameState &gamestate, const MunchkinCard &card)

  processes card type item buff
- CardTypeRetVal cardtypeaction::lvlUp (GameState &gamestate, const MunchkinCard &card)

  processes card type level up

7.10.1 Detailed Description

Class to define the functions for munchkin card types.

Author

Benjamin Lueben

In this class all the functions for the different munchkin card types are defined which are used to implement the game logic of the munchkin cardgame.

7.10.2 Typedef Documentation

7.10.2.1 CardTypeFunc

```
using CardTypeFunc = std::function<CardTypeRetVal(GameState&, const MunchkinCard&)>
```

7.11 common.h File Reference

class for common used tools, objects, etc

```
#include <array>
#include <sstream>
```

Classes

- struct testOutput

  struct for object that is used for the YOLO tests.

Namespaces

- munch_tut

  namespace for munchkin tutorial

Typedefs

- using munch_tut::Color = std::array< int, 3 >

7.11.1 Detailed Description

class for common used tools, objects, etc

Author

Benjamin Lueben

In this class helpfull things like own namespaces or custom structs are defined that can be used throughout the whole classes.

7.12 DetectionVariants.cpp File Reference

```
#include <DetectionVariants.h>
```

Functions

- int YOLOMatching (Mat &frame, const vector< String > &classes, float confThreshold, float nmsThreshold, GameState &gamestate, const vector< MunchkinCard > &cards, int modus, testOutput &fileContent)

  Function to match Munchkincards via YOLO Neuronal Network. This function is based on online Tutorial by Sunita Nayak (https://learnopencv.com/deep-learning-based-object-detection-using-yolov3-with-openc
- int ArucoMatching (Mat &frame, GameState &gamestate, Ptr< aruco::Dictionary > markerDictionary, float arucoSquareDimensions, Mat cameraMatrix, Mat distanceCoefficients, vector< MunchkinCard > cards)

  function to use Aruco Markers as workaround for matching method. Part of practical project.

7.12.1 Function Documentation

7.12.1.1 ArucoMatching()

```
int ArucoMatching (
            Mat & frame,
            GameState & gamestate,
            Ptr< aruco::Dictionary > markerDictionary,
            float arucoSquareDimensions,
            Mat cameraMatrix,
            Mat distanceCoefficients,
            vector< MunchkinCard > cards )
```

function to use Aruco Markers as workaround for matching method. Part of practical project.

Parameters

frame	current frame of webcam input
gamestate	gamestate object which stores all necessary data for tutorial
markerDictionary	aruco dictionary object of what aruco dictionary is used and needs to be detected
arucoSquareDimensions	dimension of aruco square that is printed onto the cards. Needs to be meassured.
cameraMatrix	mat object of camera matrix which stores camera calibration needed for aruco markers to function
distanceCoefficients	mat object of camera matrix which stores distance coefficients needed for aruco markers to function

Returns

1 if nothing failed.

7.12.1.2 YOLOMatching()

```
int YOLOMatching (
            Mat & frame,
            const vector< String > & classes,
            float confThreshold,
            float nmsThreshold,
            GameState & gamestate,
            const vector< MunchkinCard > & cards,
            int modus,
            testOutput & fileContent )
```

Function to match Munchkincards via YOLO Neuronal Network. This function is based on online Tutorial by Sunita Nayak (https://learnopencv.com/deep-learning-based-object-detection-using-yolov3-with-ope

Parameters

frame	current frame of webcam input
classes	vector of class names that can be found by YOLO net
confThreshold	threshold for confidence with which matches should be declared as valid/correct.
nmsThreshold	Threshold for non-maximum suppression method to remove overlapping bounding boxes
gamestate	gamestate object which stores all necessary data for tutorial
cards	vector of munchkin cards object in which all cards are stored that can be detected with the tutorial.
modus	modus for yolo matching (1 => live tutorial; 2 => testing with class name and confidence displayed; 4 => testing withour class name and confidence displayed)
fileContent	object of testOutput to store all data needed for testing

Returns

1 if nothing failed.

7.13 DetectionVariants.h File Reference

multiple Detection Variants for Munchkin Tutorial

```
#include "opencv2/aruco.hpp"
#include "opencv2/dnn.hpp"
#include "GameState.h"
#include "MunchkinCards.h"
#include <PostProcessFunctions.h>
#include <common.h>
```

Functions

- int YOLOMatching (Mat &frame, const vector< String > &classes, float confThreshold, float nmsThreshold, GameState &gamestate, const vector< MunchkinCard > &cards, int modus, testOutput &fileContent)

  Function to match Munchkincards via YOLO Neuronal Network. This function is based on online Tutorial by Sunita Nayak (https://learnopencv.com/deep-learning-based-object-detection-using-yolov3-with-openc
- int ArucoMatching (Mat &frame, GameState &gamestate, Ptr< aruco::Dictionary > markerDictionary, float arucoSquareDimensions, Mat cameraMatrix, Mat distanceCoefficients, vector< MunchkinCard > cards)

  function to use Aruco Markers as workaround for matching method. Part of practical project.

7.13.1 Detailed Description

multiple Detection Variants for Munchkin Tutorial

Author

Benjamin Lueben

In this class multiple Detection Variants are implemented to detect Munchkin Cards. Detection Variants contain Aruco Markers, Template Matching and neuronal net matching with YOLO.

7.13.2 Function Documentation

7.13.2.1 ArucoMatching()

```
int ArucoMatching (
            Mat & frame,
            GameState & gamestate,
            Ptr< aruco::Dictionary > markerDictionary,
            float arucoSquareDimensions,
            Mat cameraMatrix,
            Mat distanceCoefficients,
            vector< MunchkinCard > cards )
```

function to use Aruco Markers as workaround for matching method. Part of practical project.

Parameters

frame	current frame of webcam input
gamestate	gamestate object which stores all necessary data for tutorial
markerDictionary	aruco dictionary object of what aruco dictionary is used and needs to be detected
arucoSquareDimensions	dimension of aruco square that is printed onto the cards. Needs to be meassured.
cameraMatrix	mat object of camera matrix which stores camera calibration needed for aruco markers to function
distanceCoefficients	mat object of camera matrix which stores distance coefficients needed for aruco markers to function
cards	vector of munchkin cards object in which all cards are stored that can be detected with the tutorial.

Returns

1 if nothing failed.

7.13.2.2 YOLOMatching()

```
int YOLOMatching (
            Mat & frame,
            const vector< String > & classes,
            float confThreshold,
            float nmsThreshold,
            GameState & gamestate,
            const vector< MunchkinCard > & cards,
            int modus,
            testOutput & fileContent )
```

Function to match Munchkincards via YOLO Neuronal Network. This function is based on online Tutorial by Sunita Nayak (https://learnopencv.com/deep-learning-based-object-detection-using-yolov3-with-ope

Parameters

frame	current frame of webcam input
classes	vector of class names that can be found by YOLO net
confThreshold	threshold for confidence with which matches should be declared as valid/correct.
nmsThreshold	Threshold for non-maximum suppression method to remove overlapping bounding boxes
gamestate	gamestate object which stores all necessary data for tutorial
cards	vector of munchkin cards object in which all cards are stored that can be detected with the tutorial.
modus	modus for yolo matching (1 => live tutorial; 2 => testing with class name and confidence displayed; 4 => testing withour class name and confidence displayed)
fileContent	object of testOutput to store all data needed for testing

Returns

1 if nothing failed.

7.14 Extras.cpp File Reference

```
#include "Extras.h"
#include "GameState.h"
#include "MunchkinCards.h"
```

7.15 Extras.h File Reference

```
#include <functional>
```

Classes

- struct ExtrasRetVal

Namespaces

- extras

Typedefs

- using ExtrasFunc = std::function< ExtrasRetVal(GameState &, const MunchkinCard &)>

7.15.1 Typedef Documentation

7.15.1.1 ExtrasFunc

```
using ExtrasFunc = std::function<ExtrasRetVal(GameState&, const MunchkinCard&)>
```

7.16 GameState.h File Reference

GameState class to store all needed parameters for tutorial.

```
#include "opencv2/opencv.hpp"
#include "opencv2/imgcodecs.hpp"
#include "opencv2/imgproc.hpp"
#include <string>
#include <vector>
#include "Button.h"
```

Classes

- struct MouseParams

  Holds data of a mouse event.
- struct NNresult

  struct that holds custom result from YOLO Neuronal Net
- struct PlayerStats

  Holds data of player stats.
- struct GameState

Enumerations

- enum class MouseEvent { lclick , rclick , move , none }

  Specifies the mouse event.

7.16.1 Detailed Description

GameState class to store all needed parameters for tutorial.

Author

Benjamin Lueben

In this class all the parameters that are needed for the tutorial are stored.

7.16.2 Enumeration Type Documentation

7.16.2.1 MouseEvent

```
enum MouseEvent  [strong]
```

Specifies the mouse event.

Enumerator

lclick	left click
rclick	right click
move	mouse move
none	empty event

7.17 InputEvent.h File Reference

Class for all input events of the player through mouseklicks.

Classes

- struct InputEvent

Enumerations

- enum class EventType {
  LmPress , LmRelease , RmPress , RmRelease ,
  KeyPress , KeyRelease }

  enum of possible event types

7.17.1 Detailed Description

Class for all input events of the player through mouseklicks.

Author

Benjamin Lueben

7.17.2 Enumeration Type Documentation

7.17.2.1 EventType

```
enum EventType  [strong]
```

enum of possible event types

Enumerator

LmPress	left mouse press
LmRelease	left mouse release
RmPress	right mouse press
RmRelease	right mouse release
KeyPress	keyboard key press
KeyRelease	keyboard key release

7.18 main.cpp File Reference

```
#include "opencv2/opencv.hpp"
#include "opencv2/imgcodecs.hpp"
#include "opencv2/imgproc.hpp"
#include "opencv2/highgui.hpp"
#include "opencv2/aruco.hpp"
#include "opencv2/calib3d.hpp"
#include "opencv2/video.hpp"
#include "opencv2/dnn.hpp"
#include "opencv2/videoio.hpp"
#include "GameState.h"
#include "MunchkinCards.h"
#include "CardTypeActions.h"
#include "common.h"
#include "Button.h"
#include "ArucoGenerator.h"
#include "CameraCalibration.h"
#include "DetectionVariants.h"
#include "YOLODataGenerator.h"
#include <sstream>
```

```
#include <iostream>
#include <fstream>
#include <string>
#include <stdexcept>
#include <unordered_map>
#include <functional>
#include <random>
#include <map>
#include <iomanip>
#include <cmath>
#include <type_traits>
```

Functions

- void button_exit (const Button &b)
- void button_continue (const Button &b)
- void button_end_turn (const Button &b)
- void logic ()
- void callBackFunction (int event, int x, int y, int flags, void ∗userdata)
- int TestYOLOMatching ()
- int startWebcamMonitoring (const Mat &cameraMatrix, const Mat &distanceCoefficients, float arucoSquare↩ Dimensions)
- int testArucoMatching (const Mat &cameraMatrix, const Mat &distanceCoefficients, float arucoSquare↩ Dimensions)
- int main (int argv, char ∗∗argc)

Variables

- const float calbirationSquareDimension = 0.025f
- const float arucoSquareDimension = 0.02f
- const Size chessboardDimensions = Size(6, 9)
- float confThreshold = 0.7
- float nmsThreshold = 0.4
- int modus = 2
- int debugSingleOutput = 5
- bool multipleTests = false
- string YoloName = "40-20k"
- string MunchkinClassesNum = "40"
- string testNum = "3"
- vector< String > classes
- vector< MunchkinCard > cards
- std::unordered_map< CardType, CardTypeFunc > functMap
- GameState gamestate

7.18.1 Function Documentation

7.18.1.1 button_continue()

```
void button_continue (
            const Button & b )
```

7.18.1.2 button_end_turn()

```
void button_end_turn (
            const Button & b )
```

7.18.1.3 button_exit()

```
void button_exit (
            const Button & b )
```

7.18.1.4 callBackFunction()

```
void callBackFunction (
            int event,
            int x,
            int y,
            int flags,
            void * userdata )
```

7.18.1.5 logic()

```
void logic ( )
```

7.18.1.6 main()

```
int main (
            int argv,
            char ** argc )
```

7.18.1.7 startWebcamMonitoring()

```
int startWebcamMonitoring (
            const Mat & cameraMatrix,
            const Mat & distanceCoefficients,
            float arucoSquareDimensions )
```

7.18.1.8 testArucoMatching()

```
int testArucoMatching (
            const Mat & cameraMatrix,
            const Mat & distanceCoefficients,
            float arucoSquareDimensions )
```

7.18.1.9 TestYOLOMatching()

```
int TestYOLOMatching ( )
```

7.18.2 Variable Documentation

7.18.2.1 arucoSquareDimension

```
const float arucoSquareDimension = 0.02f
```

7.18.2.2 calbirationSquareDimension

```
const float calbirationSquareDimension = 0.025f
```

7.18.2.3 cards

```
vector<MunchkinCard> cards
```

7.18.2.4 chessboardDimensions

```
const Size chessboardDimensions = Size(6, 9)
```

7.18.2.5 classes

```
vector<String> classes
```

7.18.2.6 confThreshold

```
float confThreshold = 0.7
```

7.18.2.7 debugSingleOutput

```
int debugSingleOutput = 5
```

7.18.2.8 functMap

```
std::unordered_map<CardType, CardTypeFunc> functMap
```

7.18.2.9 gamestate

```
GameState gamestate
```

7.18.2.10 modus

```
int modus = 2
```

7.18.2.11 multipleTests

```
bool multipleTests = false
```

7.18.2.12 MunchkinClassesNum

```
string MunchkinClassesNum = "40"
```

7.18.2.13 nmsThreshold

```
float nmsThreshold = 0.4
```

7.18.2.14 testNum

```
string testNum = "3"
```

7.18.2.15 YoloName

```
string YoloName = "40-20k"
```

7.19 MunchkinCards.cpp File Reference

```
#include "MunchkinCards.h"
```

7.20 MunchkinCards.h File Reference

MunchkinCards class in which all parameters for the munchkin cards are stored.

```
#include <string>
#include <vector>
#include "opencv2/opencv.hpp"
#include "opencv2/imgcodecs.hpp"
#include "opencv2/imgproc.hpp"
#include <functional>
#include "GameState.h"
#include "BadThings.h"
```

Classes

- class MunchkinCard

  class of munchkin cards with params that could be needed for a single card

Enumerations

- enum class ParentCardType { door , treasure }

  enum class for parent card types
- enum class CardType {
  curse , munchClass , joker , lvlUp ,
  itemBuff , monster , race , item ,
  removeCard }

  enum class for "child" card Types
- enum class ItemType {
  armor , shoes , hat , boni ,
  weapon , joker , clothing }

  enum class of item types

7.20.1 Detailed Description

MunchkinCards class in which all parameters for the munchkin cards are stored.

Author

Benjamin Lueben

In this class the munchkin Card object is defined with every parameter the card could have and with custom constructors which could be used to define the cards more easily

7.20.2 Enumeration Type Documentation

7.20.2.1 CardType

```
enum CardType  [strong]
```

enum class for "child" card Types

Enumerator

curse	card type curse
munchClass	card type munchkin class
joker	card type joker
lvlUp	card type level up
itemBuff	card type item buff
monster	card type monster
race	card type munchkin race
item	card type item
removeCard	param to signal removing a card

7.20.2.2 ItemType

```
enum ItemType  [strong]
```

enum class of item types

Enumerator

armor	item type armor
shoes	item class shoes
hat	item class hat
boni	item class boni
weapon	item class weapon
joker	item class joker
clothing	item class clothing

7.20.2.3 ParentCardType

```
enum ParentCardType  [strong]
```

enum class for parent card types

Enumerator

door	card type door
treasure	card type treasure

7.21 PostProcessFunctions.cpp File Reference

```
#include <PostProcessFunctions.h>
#include <algorithm>
#include <vector>
#include <cmath>
```

Functions

- Color chooseCardColor (int markerId, vector< MunchkinCard > cards)

  function to choose color e.g for boundingBoxes that is bound to type of single munchkin card.
- vector< String > getOutputsNames (const cv::dnn::Net &net)

  function to get Outputs names from neuronal net. This function is based on online Tutorial by Sunita Nayak (https://learnopencv.com/deep-learning-based-object-detection-using-yolov3-with-opencv-pyth
- void drawPred (int classId, float conf, int left, int top, int right, int bottom, Mat &frame, vector< String > classes, vector< MunchkinCard > cards, int modus)

function to draw prediction boxes for YOLO matching. This function is based on online Tutorial by Sunita Nayak (https://learnopencv.com/deep-learning-based-object-detection-using-yolov3-with-opencv-pyth

- void postprocess (Mat &frame, const vector< Mat > &outs, float confThreshold, float nmsThreshold, GameState &gamestate, const vector< String > &classes, const vector< MunchkinCard > &cards, int modus, testOutput &fileContent)

  function which handles the post processing of outputs of YOLO matching method. This function is based on online Tutorial by Sunita Nayak (https://learnopencv.com/deep-learning-based-object-detection-using-yolov3-

7.21.1 Function Documentation

7.21.1.1 chooseCardColor()

```
Color chooseCardColor (
            int markerId,
            vector< MunchkinCard > cards )
```

function to choose color e.g for boundingBoxes that is bound to type of single munchkin card.

Parameters

markerId	id of munchkin card that was detected by tutorial. (former showed marker id which was referenced with munchkin card)
cards	vector of munchkin card objects with all munchkin cards that are currently usable with the tutorial.

Returns

cv::Color object with color for card (BGR)

7.21.1.2 drawPred()

```
void drawPred (
            int classId,
            float conf,
            int left,
            int top,
            int right,
            int bottom,
            Mat & frame,
            vector< String > classes,
            vector< MunchkinCard > cards,
            int modus )
```

function to draw prediction boxes for YOLO matching. This function is based on online Tutorial by Sunita Nayak (https://learnopencv.com/deep-learning-based-object-detection-using-yolov3-with-opencv-py

Parameters

classId	class id of found class
conf	confidence of class for predicted box
left	x value of top-left corner of bounding box to be drawn
top	width of bounding box to be drawn
right	y value of top-left corner of bounding box to be drawn
bottom	height of bounding box to be drawn
frame	Mat obejct of current frame of webcam input
classes	vector of classes name that can be found with YOLO net
cards	vector ob MunchkinCards object which stores all current detectable munchkin cards
modus	modus for yolo matching (1 => live tutorial; 2 => testing with class name and confidence displayed; 4 => testing withour class name and confidence displayed)

7.21.1.3 getOutputsNames()

```
vector<String> getOutputsNames (
            const cv::dnn::Net & net )
```

function to get Outputs names from neuronal net. This function is based on online Tutorial by Sunita Nayak (https://learnopencv.com/deep-learning-based-object-detection-using-yolov3-with-opencv-py

Parameters

net	cv::dnn::Net reference of neuronal net that is used in tutorial.

Returns

array of names that can be found by neuronal net.

7.21.1.4 postprocess()

```
void postprocess (
            Mat & frame,
            const vector< Mat > & outs,
            float confThreshold,
            float nmsThreshold,
            GameState & gamestate,
            const vector< String > & classes,
            const vector< MunchkinCard > & cards,
            int modus,
            testOutput & fileContent )
```

function which handles the post processing of outputs of YOLO matching method. This function is based on online Tutorial by Sunita Nayak (https://learnopencv.com/deep-learning-based-object-detection-using-yol

Parameters

frame	Mat obejct of current frame of webcam input
outs	vector of mat objects in which outputs from YOLO net are stored in.
confThreshold	threshold for confidence with which matches should be declared as valid/correct.
nmsThreshold	Threshold for non-maximum suppression method to remove overlapping bounding boxes
gamestate	gamestate object which stores all necessary data for tutorial
cards	vector of munchkin cards object in which all cards are stored that can be detected with the tutorial.
classes	vector of classes name that can be found with YOLO net
modus	modus for yolo matching (1 => live tutorial; 2 => testing with class name and confidence displayed; 4 => testing withour class name and confidence displayed)
fileContent	object of testOutput to store all data needed for testing

7.22 PostProcessFunctions.h File Reference

all util/post process functions

```
#include <MunchkinCards.h>
#include <GameState.h>
#include <common.h>
```

Namespaces

- constants

Functions

- Color chooseCardColor (int markerId, vector< MunchkinCard > cards)

  function to choose color e.g for boundingBoxes that is bound to type of single munchkin card.
- vector< String > getOutputsNames (const cv::dnn::Net &net)

  function to get Outputs names from neuronal net. This function is based on online Tutorial by Sunita Nayak (https://learnopencv.com/deep-learning-based-object-detection-using-yolov3-with-opencv-pyth
- void drawPred (int classId, float conf, int left, int top, int right, int bottom, Mat &frame, vector< String > classes, vector< MunchkinCard > cards, int modus)

  function to draw prediction boxes for YOLO matching. This function is based on online Tutorial by Sunita Nayak (https://learnopencv.com/deep-learning-based-object-detection-using-yolov3-with-opencv-pyth
- void postprocess (Mat &frame, const vector< Mat > &outs, float confThreshold, float nmsThreshold, GameState &gamestate, const vector< String > &classes, const vector< MunchkinCard > &cards, int modus, testOutput &fileContent)

  function which handles the post processing of outputs of YOLO matching method. This function is based on online Tutorial by Sunita Nayak (https://learnopencv.com/deep-learning-based-object-detection-using-yolov3-

Variables

- constexpr int constants::yolo_grid_size = 13

7.22.1 Detailed Description

all util/post process functions

Author

Benjamin Lueben

In this class all the needed util and post process functions for Detection Variants are defined.

7.22.2 Function Documentation

7.22.2.1 chooseCardColor()

```
Color chooseCardColor (
            int markerId,
            vector< MunchkinCard > cards )
```

function to choose color e.g for boundingBoxes that is bound to type of single munchkin card.

Parameters

markerId	id of munchkin card that was detected by tutorial. (former showed marker id which was referenced with munchkin card)
cards	vector of munchkin card objects with all munchkin cards that are currently usable with the tutorial.

Returns

cv::Color object with color for card (BGR)

7.22.2.2 drawPred()

```
void drawPred (
            int classId,
            float conf,
            int left,
            int top,
            int right,
            int bottom,
            Mat & frame,
            vector< String > classes,
            vector< MunchkinCard > cards,
            int modus )
```

function to draw prediction boxes for YOLO matching. This function is based on online Tutorial by Sunita Nayak (https://learnopencv.com/deep-learning-based-object-detection-using-yolov3-with-opencv-py

Parameters

classId	class id of found class
conf	confidence of class for predicted box
left	x value of top-left corner of bounding box to be drawn
top	width of bounding box to be drawn
right	y value of top-left corner of bounding box to be drawn
bottom	height of bounding box to be drawn
frame	Mat obejct of current frame of webcam input
classes	vector of classes name that can be found with YOLO net
cards	vector ob MunchkinCards object which stores all current detectable munchkin cards
modus	modus for yolo matching (1 => live tutorial; 2 => testing with class name and confidence displayed; 4 => testing withour class name and confidence displayed)

7.22.2.3 getOutputsNames()

```
vector<String> getOutputsNames (
            const cv::dnn::Net & net )
```

function to get Outputs names from neuronal net. This function is based on online Tutorial by Sunita Nayak (https://learnopencv.com/deep-learning-based-object-detection-using-yolov3-with-opencv-py

Parameters

net	cv::dnn::Net reference of neuronal net that is used in tutorial.

Returns

array of names that can be found by neuronal net.

7.22.2.4 postprocess()

```
void postprocess (
            Mat & frame,
            const vector< Mat > & outs,
            float confThreshold,
            float nmsThreshold,
            GameState & gamestate,
            const vector< String > & classes,
            const vector< MunchkinCard > & cards,
            int modus,
            testOutput & fileContent )
```

function which handles the post processing of outputs of YOLO matching method. This function is based on online Tutorial by Sunita Nayak (https://learnopencv.com/deep-learning-based-object-detection-using-yol

Parameters

frame	Mat obejct of current frame of webcam input
outs	vector of mat objects in which outputs from YOLO net are stored in.
confThreshold	threshold for confidence with which matches should be declared as valid/correct.
nmsThreshold	Threshold for non-maximum suppression method to remove overlapping bounding boxes
gamestate	gamestate object which stores all necessary data for tutorial
cards	vector of munchkin cards object in which all cards are stored that can be detected with the tutorial.
classes	vector of classes name that can be found with YOLO net
modus	modus for yolo matching (1 => live tutorial; 2 => testing with class name and confidence displayed; 4 => testing withour class name and confidence displayed)
fileContent	object of testOutput to store all data needed for testing

7.23 README.md File Reference

7.24 TTMunchkinTut.cpp File Reference

7.25 YOLODataGenerator.cpp File Reference

```
#include <YOLODataGenerator.h>
#include "opencv2/opencv.hpp"
#include "opencv2/imgcodecs.hpp"
#include "opencv2/imgproc.hpp"
#include "opencv2/highgui.hpp"
```

Functions

- template<typename T >
  T unirand (T a, T b)
- template<typename T >
  T uninorm (T mean, T sdev)
- int generateDatasetNN ()

  generates datasets which were used to train YOLO nets. creates pictures and txt files with coordinates of bounding boxes relativ to picture size. values num_images and num_inputPictures are used for how many images and how many input pictures should be used. num_inputPictures means how many of loaded munchkin cards should be used.

7.25.1 Function Documentation

7.25.1.1 generateDatasetNN()

```
int generateDatasetNN ( )
```

generates datasets which were used to train YOLO nets. creates pictures and txt files with coordinates of bounding boxes relativ to picture size. values num_images and num_inputPictures are used for how many images and how many input pictures should be used. num_inputPictures means how many of loaded munchkin cards should be used.

Returns

1 when finished

7.25.1.2 uninorm()

```
template<typename T >
T uninorm (
            T mean,
            T sdev )
```

7.25.1.3 unirand()

```
template<typename T >
T unirand (
            T a,
            T b )
```

7.26 YOLODataGenerator.h File Reference

generates Data for YOLO training

```
#include <vector>
#include <sstream>
#include <iostream>
#include <fstream>
#include <string>
#include <stdexcept>
#include <unordered_map>
#include <functional>
#include <random>
#include <map>
#include <iomanip>
#include <cmath>
#include <type_traits>
```

Functions

- template<typename T >
  T unirand (T a, T b)
- template<typename T >
  T uninorm (T mean, T sdev)
- int generateDatasetNN ()

  generates datasets which were used to train YOLO nets. creates pictures and txt files with coordinates of bounding boxes relativ to picture size. values num_images and num_inputPictures are used for how many images and how many input pictures should be used. num_inputPictures means how many of loaded munchkin cards should be used.

7.26.1 Detailed Description

generates Data for YOLO training

Author

Benjamin Lueben

In this class the functions needed for generating Data for YOLO neural net are defined.

7.26.2 Function Documentation

7.26.2.1 generateDatasetNN()

```
int generateDatasetNN ( )
```

generates datasets which were used to train YOLO nets. creates pictures and txt files with coordinates of bounding boxes relativ to picture size. values num_images and num_inputPictures are used for how many images and how many input pictures should be used. num_inputPictures means how many of loaded munchkin cards should be used.

Returns

1 when finished

7.26.2.2 uninorm()

```
template<typename T >
T uninorm (
            T mean,
            T sdev )
```

7.26.2.3 unirand()

```
template<typename T >
T unirand (
            T a,
            T b )
```

Index