Ссылка на презентацию: https://drive.google.com/file/d/1ZrHQP0q-ihuX9N9lkD4FXf-yfplKKYqL/view?usp=sharing

Проект по формированию монетизации игры "Космические братья"

Постановка задачи

Цель проекта:

- Сформировать модель монетизации игры.

Задачи проекта:

- Найти оптимальное время для запуска рекламы.
- Найти лучший рекламный источник по соотношению цена/качество.
- Дать рекомендации создателям по улучшению UX.

Этап 1: Выгрузка и изучение данных

Выгрузить три датасета *ad_costs*, *game_actions*, *user_source* с данными о рекламе, внутриигровых действиях и пользователях.

Путь к файлу *ad_costs*: /datasets/ad_costs.csv

Путь к файлу *game_actions*: /datasets/game_actions.csv

Путь к файлу *user_source*: /datasets/user_source.csv

Изучить общую информацию о датасетах, какого типа данные представлены и в каких количествах.

Этап 2: Предобработка данных

Подготовить данные к анализу:

- Исследовать пропущенные значения.
- Исследовать соответствие типов.
- Исследовать дубликаты.
- Проверить корректность наименований колонок.
- Переименовать колонки.
- Удалить дубликаты.
- Привести типы.
- Заменить пропущенные значения.

Этап 3: Исследовательский анализ данных

- Составить рейтинг рекламных источников по стоимости за клики.
- Найти самый дорогой и самый дешевый источник на основе рейтинга.
- Составить рейтинг рекламных исчтоников по количеству привлеченных пользователей.
- Найти самый продуктивный и наименее продуктивный источник на основе рейтинга.
- Составить рейтинг поплуряности событий по количеству.
- Составить рейтинг популярности строящихся зданий.
- Составить общий рейтинг рекламных источников.

Этап 4: Анализ влияния событий на совершение целевого события

Главной задачей создаталей игры является окуп к окончанию первого уровня, поэтому проанализируем события, предшествующие переходу на новый уровень. Учтем, что показ рекламы предлагается во время строительства.

- Разбить пользователей на категории в зависимости от их способа перехода на второй уровень и сравнить их соотношение.
- Рассчитать сколько времени пользователи тратят на достижение второго уровня с момента регистрации для каждой из категорий.
- Изучить действия пользователей до достижения finished_stage_1. Сколько действий совершают игроки для перехода на новый уровень?
- Рассчитать сколько пользователей и на каких этапах останавливаются, так и не совершив переход на новый уровень.
- Составить рейтинг источников по категориям игроков.

Этап 5: Проверка статистических гипотез

- Проверить гипотезу различия времени прохождения уровня между пользователями, которые заканчивают уровень через реализацию проекта, и пользователями, которые заканчивают уровень победой над другим игроком.
- Проверить гипотезу о различии трат за клики на Инстаграм и Фэйсбук

Этап 6: Подведение итогов и презентация результатов

Основная монетизация игры только планируется, предполагается показ рекламы во время строительства. На основе посчитанных рейтингов и сводных таблиц мы определим, когда лучше всего показывать рекламу и какой категории пользователей. Если большая часть игроков проходит уровень с помощью битвы, то изначальная идея показывать рекламу во время строительства провальная и нужно учесть соотношение игроков в каждой из категорий.

- Определить модель монетизации на основе полученных исследований и подберем лучший вариант показа рекламы.
- Подвести итоги и дать рекомендации по монетазации для создателей игры, а также подготовить презентацию исследования.
- Сформулировать основные тезисы, отметить важные наблюдения и предоставить результаты в виде презентации, дополнить ее графиками и сводными таблицами.
- Приложить ссылку на презентацию в формате PDF.

Этап 7: Построение дашбордов

1. Набор №1
   A. Построить диаграмму распределения количества построенных объектов
   B. Добавить индикатор количества пользователей
   C. Добавить фильтр дашборда по признаку завершения уровня

1. Набор №2
   A. Построить диаграмму, отображающую количество событий по периодам
   B. Построить гистограмму, отображающую количество пользователей, пришедших из разных источников
   C. Добавить фильтр дашборда по типу события

Выгрузка и изучение данных

In [6]:
```
#импортируем нужные библиотеки
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np
import seaborn as sns
import plotly.express as px
from datetime import datetime, timedelta
import datetime as dt
from math import factorial
from scipy import stats as st
from plotly import graph_objects as go
```

In [7]:
```
#выгружаем данные с помощью конструкции try except
try:
    ad_costs = pd.read_csv('ad_costs.csv')
except FileNotFoundError:
    print("Файл не найден.")
except pd.errors.EmptyDataError:
    print("Данные пусты")

try:
    game_actions = pd.read_csv('game_actions.csv')
except FileNotFoundError:
    print("Файл не найден.")
except pd.errors.EmptyDataError:
    print("Данные пусты")

try:
    user_source = pd.read_csv('user_source.csv')
except FileNotFoundError:
    print("Файл не найден.")
except pd.errors.EmptyDataError:
    print("Данные пусты")
```

In [8]:
```
ad_costs.info()
ad_costs.head()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 28 entries, 0 to 27
Data columns (total 3 columns):
 #   Column  Non-Null Count  Dtype  
---  ------  --------------  -----  
 0   source  28 non-null     object 
 1   day     28 non-null     object 
 2   cost    28 non-null     float64
dtypes: float64(1), object(2)
memory usage: 800.0+ bytes
```

Out[8]:

	source	day	cost
0	facebook_ads	2020-05-03	935.882786
1	facebook_ads	2020-05-04	548.354480
2	facebook_ads	2020-05-05	260.185754
3	facebook_ads	2020-05-06	177.982200
4	facebook_ads	2020-05-07	111.766796

В датафрейме ad_costs 3 столбца с 28 строками без пропусков.

In [9]:
```
game_actions.info()
game_actions.head()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 135640 entries, 0 to 135639
Data columns (total 5 columns):
 #   Column          Non-Null Count   Dtype 
---  ------          --------------   ----- 
 0   event_datetime  135640 non-null  object
 1   event           135640 non-null  object
 2   building_type   127957 non-null  object
 3   user_id         135640 non-null  object
 4   project_type    1866 non-null    object
dtypes: object(5)
memory usage: 5.2+ MB
```

Out[9]:

	event_datetime	event	building_type	user_id	project_type
0	2020-05-04 00:00:01	building	assembly_shop	55e92310-cb8e-4754-b622-597e124b03de	NaN
1	2020-05-04 00:00:03	building	assembly_shop	c07b1c10-f477-44dc-81dc-ec82254b1347	NaN
2	2020-05-04 00:00:16	building	assembly_shop	6edd42cc-e753-4ff6-a947-2107cd560710	NaN
3	2020-05-04 00:00:16	building	assembly_shop	92c69003-d60a-444a-827f-8cc51bf6bf4c	NaN
4	2020-05-04 00:00:35	building	assembly_shop	cdc6bb92-0ccb-4490-9866-ef142f09139d	NaN

В датафрейме game_actions 5 столбцов с 135640 строками. В 2 столбцах есть пропуски.

In [10]:
```
user_source.info()
user_source.head()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 13576 entries, 0 to 13575
Data columns (total 2 columns):
 #   Column   Non-Null Count  Dtype 
---  ------   --------------  ----- 
 0   user_id  13576 non-null  object
 1   source   13576 non-null  object
dtypes: object(2)
memory usage: 212.2+ KB
```

Out[10]:

	user_id	source
0	0001f83c-c6ac-4621-b7f0-8a28b283ac30	facebook_ads
1	00151b4f-ba38-44a8-a650-d7cf130a0105	yandex_direct
2	001aaea6-3d14-43f1-8ca8-7f48820f17aa	youtube_channel_reklama
3	001d39dc-366c-4021-9604-6a3b9ff01e25	instagram_new_adverts
4	002f508f-67b6-479f-814b-b05f00d4e995	facebook_ads

В датафрейме user_source 2 столбца с 13576 строками без пропусков.

Предобработка данных

Предобработка ad_costs

In [11]:
```
display(ad_costs)
ad_costs.info()
```

	source	day	cost
0	facebook_ads	2020-05-03	935.882786
1	facebook_ads	2020-05-04	548.354480
2	facebook_ads	2020-05-05	260.185754
3	facebook_ads	2020-05-06	177.982200
4	facebook_ads	2020-05-07	111.766796
5	facebook_ads	2020-05-08	68.009276
6	facebook_ads	2020-05-09	38.723350
7	instagram_new_adverts	2020-05-03	943.204717
8	instagram_new_adverts	2020-05-04	502.925451
9	instagram_new_adverts	2020-05-05	313.970984
10	instagram_new_adverts	2020-05-06	173.071145
11	instagram_new_adverts	2020-05-07	109.915254
12	instagram_new_adverts	2020-05-08	71.578739
13	instagram_new_adverts	2020-05-09	46.775400
14	yandex_direct	2020-05-03	969.139394
15	yandex_direct	2020-05-04	554.651494
16	yandex_direct	2020-05-05	308.232990
17	yandex_direct	2020-05-06	180.917099
18	yandex_direct	2020-05-07	114.429338
19	yandex_direct	2020-05-08	62.961630
20	yandex_direct	2020-05-09	42.779505
21	youtube_channel_reklama	2020-05-03	454.224943
22	youtube_channel_reklama	2020-05-04	259.073224
23	youtube_channel_reklama	2020-05-05	147.041741
24	youtube_channel_reklama	2020-05-06	88.506074
25	youtube_channel_reklama	2020-05-07	55.740645
26	youtube_channel_reklama	2020-05-08	40.217907
27	youtube_channel_reklama	2020-05-09	23.314669

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 28 entries, 0 to 27
Data columns (total 3 columns):
 #   Column  Non-Null Count  Dtype  
---  ------  --------------  -----  
 0   source  28 non-null     object 
 1   day     28 non-null     object 
 2   cost    28 non-null     float64
dtypes: float64(1), object(2)
memory usage: 800.0+ bytes
```

Каких-либо недочетов в данных нет, они достаточно чистые и с ними уже можно работать

In [12]:
```
#переводим данные с датой из типа object в тип datetime и округаляем значения
ad_costs['day'] = pd.to_datetime(ad_costs['day']).dt.date

ad_costs['cost'] = round(ad_costs['cost'],2)
```

Предобработка game_actions

In [13]:
```
display(game_actions)
game_actions.info()
```

	event_datetime	event	building_type	user_id	project_type
0	2020-05-04 00:00:01	building	assembly_shop	55e92310-cb8e-4754-b622-597e124b03de	NaN
1	2020-05-04 00:00:03	building	assembly_shop	c07b1c10-f477-44dc-81dc-ec82254b1347	NaN
2	2020-05-04 00:00:16	building	assembly_shop	6edd42cc-e753-4ff6-a947-2107cd560710	NaN
3	2020-05-04 00:00:16	building	assembly_shop	92c69003-d60a-444a-827f-8cc51bf6bf4c	NaN
4	2020-05-04 00:00:35	building	assembly_shop	cdc6bb92-0ccb-4490-9866-ef142f09139d	NaN
...	...	...	...	...	...
135635	2020-06-05 00:08:06	building	research_center	f21d179f-1c4b-437e-b9c6-ab1976907195	NaN
135636	2020-06-05 02:25:12	finished_stage_1	NaN	515c1952-99aa-4bca-a7ea-d0449eb5385a	NaN
135637	2020-06-05 08:57:52	building	research_center	ed3e7d02-8a96-4be7-9998-e9813ff9c316	NaN
135638	2020-06-05 12:12:27	finished_stage_1	NaN	32572adb-900f-4b5d-a453-1eb1e6d88d8b	NaN
135639	2020-06-05 12:32:49	finished_stage_1	NaN	f21d179f-1c4b-437e-b9c6-ab1976907195	NaN

135640 rows × 5 columns

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 135640 entries, 0 to 135639
Data columns (total 5 columns):
 #   Column          Non-Null Count   Dtype 
---  ------          --------------   ----- 
 0   event_datetime  135640 non-null  object
 1   event           135640 non-null  object
 2   building_type   127957 non-null  object
 3   user_id         135640 non-null  object
 4   project_type    1866 non-null    object
dtypes: object(5)
memory usage: 5.2+ MB
```

В данных есть проблемы. Необходимо:

- изменить тип данных в столбце с датой
- обработать пропуски в столбце building_type
- обработать пропуски в столбце project_type
- обработать дубликаты

In [14]:
```
#изменяем тип данных из object в datetime
game_actions['event_datetime'] = pd.to_datetime(game_actions['event_datetime']).dt.date
```

In [15]:
```
display(game_actions[game_actions['building_type'].isna()])
```

	event_datetime	event	building_type	user_id	project_type
6659	2020-05-04	finished_stage_1	NaN	ced7b368-818f-48f6-9461-2346de0892c5	NaN
13134	2020-05-05	finished_stage_1	NaN	7ef7fc89-2779-46ea-b328-9e5035b83af5	NaN
15274	2020-05-05	finished_stage_1	NaN	70db22b3-c2f4-43bc-94ea-51c8d2904a29	NaN
16284	2020-05-05	finished_stage_1	NaN	903fc9ef-ba97-4b12-9d5c-ac8d602fbd8b	NaN
19650	2020-05-06	finished_stage_1	NaN	58e077ba-feb1-4556-a5a0-d96bd04efa39	NaN
...	...	...	...	...	...
135632	2020-06-04	finished_stage_1	NaN	22cce310-fe10-41a2-941b-9c3d63327fea	NaN
135633	2020-06-04	finished_stage_1	NaN	d477dde8-7c22-4f23-9c4f-4ec31a1aa4c8	NaN
135636	2020-06-05	finished_stage_1	NaN	515c1952-99aa-4bca-a7ea-d0449eb5385a	NaN
135638	2020-06-05	finished_stage_1	NaN	32572adb-900f-4b5d-a453-1eb1e6d88d8b	NaN
135639	2020-06-05	finished_stage_1	NaN	f21d179f-1c4b-437e-b9c6-ab1976907195	NaN

7683 rows × 5 columns

Как мы видим пропуски в столбце building_type вызваны тем, что игроки переходят на новый уровень без строительства путем битвы. Оставим их как есть, так как эти данные важны.

In [16]:
```
display(game_actions[game_actions['project_type'].notna()])
display(game_actions['event'].value_counts())
```

	event_datetime	event	building_type	user_id	project_type
47121	2020-05-08	project	NaN	e3c66498-9d45-4000-9392-f81e6796e7da	satellite_orbital_assembly
57398	2020-05-09	project	NaN	936e7af6-8338-4703-a1df-fc6c3f5b8e34	satellite_orbital_assembly
58797	2020-05-09	project	NaN	a4491c86-c498-4f74-a56e-65c136d0e9a1	satellite_orbital_assembly
61174	2020-05-09	project	NaN	85d9e675-562b-4329-8bbd-14d3b39096be	satellite_orbital_assembly
63770	2020-05-10	project	NaN	1889ca71-3c57-4e61-9ea6-a711971bbf0a	satellite_orbital_assembly
...	...	...	...	...	...
135602	2020-06-02	project	NaN	9d98001c-7e14-40d7-896e-46b3047365fd	satellite_orbital_assembly
135603	2020-06-02	project	NaN	df4a1e13-eba9-4928-a7cf-ee303d6f80f9	satellite_orbital_assembly
135609	2020-06-02	project	NaN	82e46f34-e243-4728-8e20-2e171fc33ea4	satellite_orbital_assembly
135617	2020-06-03	project	NaN	fe032991-71e0-48c5-889f-4c3805ba4c9b	satellite_orbital_assembly
135630	2020-06-04	project	NaN	d477dde8-7c22-4f23-9c4f-4ec31a1aa4c8	satellite_orbital_assembly

1866 rows × 5 columns

```
building            127957
finished_stage_1      5817
project               1866
Name: event, dtype: int64
```

Пропуски в столбце project_type тоже не будем трогать, так как значение этого столбца напрямую зависит от значения project в столбце event. Оставим прпоуски

In [17]:
```
#удалим явные дубликаты из датафрейма
game_actions = game_actions.drop_duplicates()
```

Предобработка user_source

In [18]:
```
display(user_source)
user_source.info()
```

	user_id	source
0	0001f83c-c6ac-4621-b7f0-8a28b283ac30	facebook_ads
1	00151b4f-ba38-44a8-a650-d7cf130a0105	yandex_direct
2	001aaea6-3d14-43f1-8ca8-7f48820f17aa	youtube_channel_reklama
3	001d39dc-366c-4021-9604-6a3b9ff01e25	instagram_new_adverts
4	002f508f-67b6-479f-814b-b05f00d4e995	facebook_ads
...	...	...
13571	ffef4fed-164c-40e1-bde1-3980f76d0fb5	instagram_new_adverts
13572	fffab3da-da0e-4e30-ae62-10d0a2e24a4e	facebook_ads
13573	fffb626c-5ab6-47c9-8113-2062a2f18494	yandex_direct
13574	ffff194a-56b7-4c12-860d-3485242ae7f5	instagram_new_adverts
13575	ffff69cc-fec1-4fd3-9f98-93be1112a6b8	facebook_ads

13576 rows × 2 columns

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 13576 entries, 0 to 13575
Data columns (total 2 columns):
 #   Column   Non-Null Count  Dtype 
---  ------   --------------  ----- 
 0   user_id  13576 non-null  object
 1   source   13576 non-null  object
dtypes: object(2)
memory usage: 212.2+ KB
```

В целом данные очень чистые, однако проверим данные на неявные дубликаты

In [19]:
```
display(user_source['source'].value_counts())
```

```
yandex_direct              4817
instagram_new_adverts      3347
facebook_ads               2726
youtube_channel_reklama    2686
Name: source, dtype: int64
```

In [20]:
```
user_source = user_source.drop_duplicates()
```

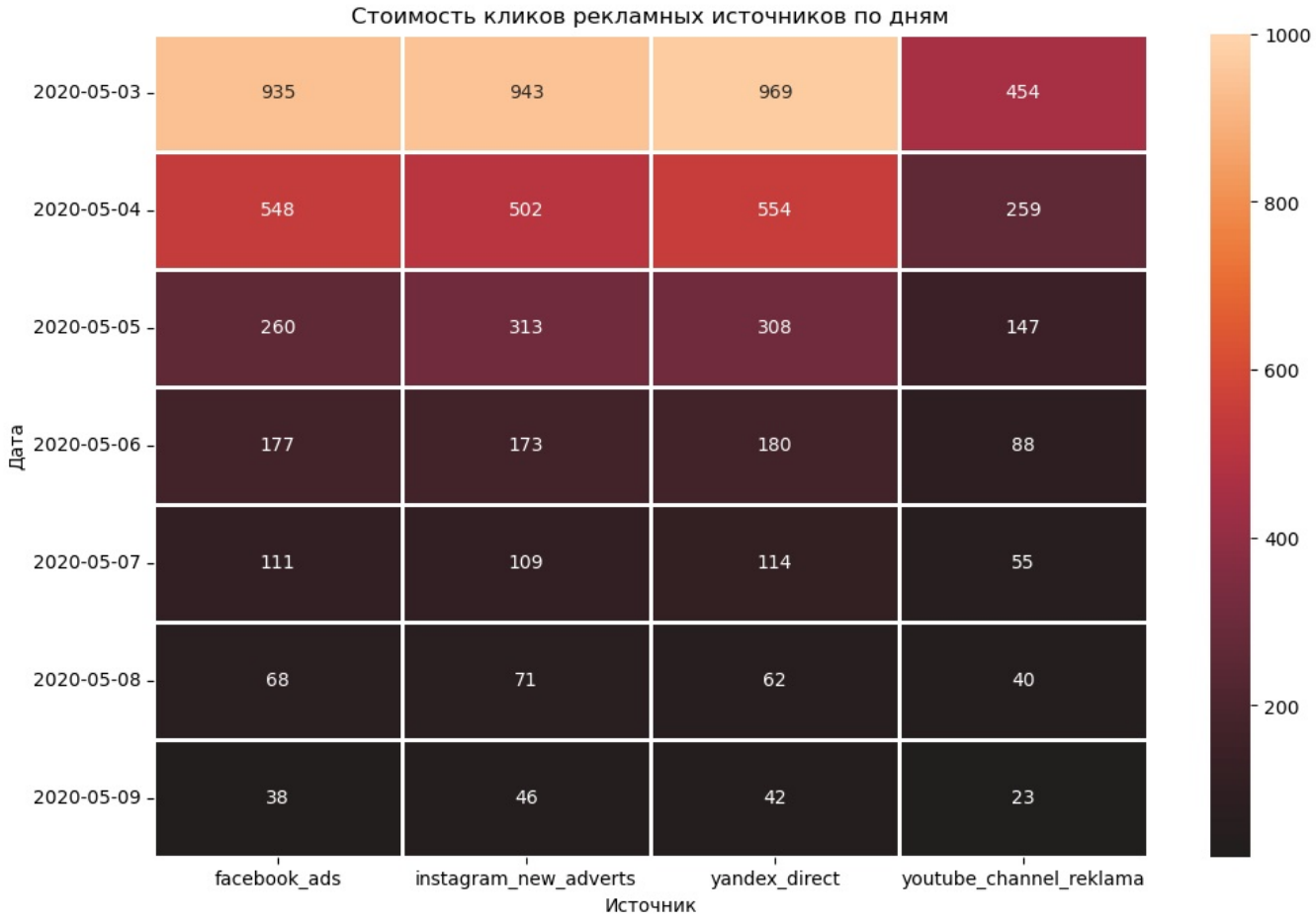
Данные обработаны и готовы к анализу. Благодарю датаинженеров за хорошую работу и предоставленные данные.

Исследовательский анализ данных

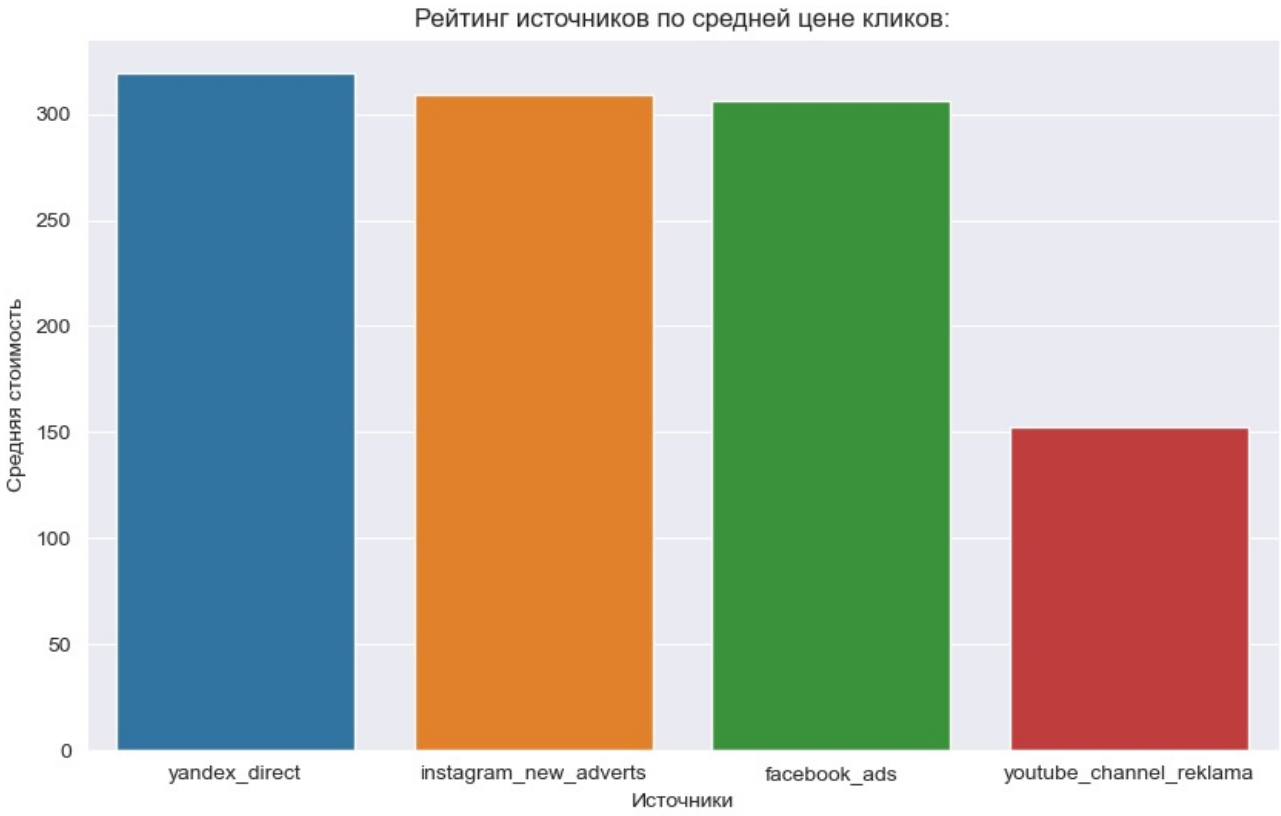
Рейтинг рекламных источников по стоимости за клики.

In [21]:
```
#создаем таблицу стоимости кликов по дням дял каждого источника
rating_cost = ad_costs.pivot_table(index = 'day', columns = 'source', values = 'cost')

#средняя стоимость кликов для каждог оисчтоника
rating_cost_mean = ad_costs.pivot_table(index = 'source',  values = 'cost', aggfunc = 'mean').reset_index().sor

display(rating_cost)

#тепловая карта стоимости кликов
plt.figure(figsize = (12,8))
rating_cost = rating_cost.astype('int64')
sns.heatmap(rating_cost, annot = True, center = 0,  fmt='',vmax = 1000, vmin = 20, linewidths = 1)
plt.xlabel('Источник')
plt.ylabel('Дата')
plt.title('Стоимость кликов рекламных источников по дням')
plt.yticks(rotation = 0)
plt.show()

display(rating_cost_mean)

#график сердней стоимости  кликов за все дни
plt.figure(figsize = (10,6))
sns.set_style("darkgrid")
sns.barplot(data = rating_cost_mean, x = 'source', y = 'cost')
plt.title('Рейтинг источников по средней цене кликов:')
plt.xlabel('Источники')
plt.ylabel('Средняя стоимость')
plt.show()
```

source day	facebook_ads	instagram_new_adverts	yandex_direct	youtube_channel_reklama
2020-05-03	935.88	943.20	969.14	454.22
2020-05-04	548.35	502.93	554.65	259.07
2020-05-05	260.19	313.97	308.23	147.04
2020-05-06	177.98	173.07	180.92	88.51
2020-05-07	111.77	109.92	114.43	55.74
2020-05-08	68.01	71.58	62.96	40.22
2020-05-09	38.72	46.78	42.78	23.31



	source	cost
0	yandex_direct	319.015714
1	instagram_new_adverts	308.778571
2	facebook_ads	305.842857
3	youtube_channel_reklama	152.587143



Как мы видим на сводной таблице и по тепловой карте, абсолютно все источники в первый день имеют пиковую стоимость пиков, после чего стоимость кликов у каждого из источников постепенно падает.

Рейтинг источников по самым дорогим кликам по дням: 1) Инстаграм

2) Яндекс

3) Инстаграм

4) Яндекс

5) Яндекс

6) Инстаграм

7) Инстаграм

Самые дешевые клики по всем дням имеет Ютуб.

Рейтинг источников по средней цене кликов: 1) Яндекс

2) Инстаграм

3) Фэйсбук

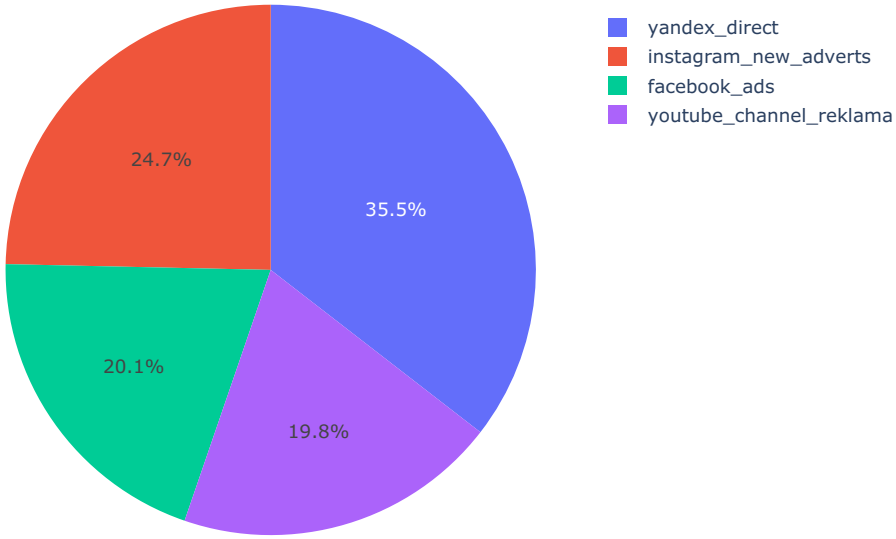
4) Ютуб

Рейтинг рекламных исчтоников по количеству привлеченных пользователей.

In [22]:
```
#таблица привлеченных пользователей по источникам
rating_source = user_source.groupby('source')['user_id'].agg('count').reset_index().sort_values(by = 'user_id',
rating_source['ratio_%'] = round(rating_source['user_id'] / rating_source['user_id'].sum() * 100, 1)
rating_source = rating_source.rename(columns = {'user_id' : 'users'})
display(rating_source)

#круговая диаграмма таблицы выше
fig = px.pie(rating_source, values='users', names = 'source', title='Рекламные исчтоники по количеству привлече
fig.show()
```

	source	users	ratio_%
0	yandex_direct	4817	35.5
1	instagram_new_adverts	3347	24.7
2	facebook_ads	2726	20.1
3	youtube_channel_reklama	2686	19.8

Рекламные исчтоники по количеству привлеченных пользователей.



Рейтинг рекламных исчтоников по количеству привлеченных пользователей: 1) Яндекс

2) Инстаграм

3) Фэйсбук

4) Ютуб

Как мы видим, рейтинг привлеченных пользователей совпадает с рейтингом источников по средней цене кликов

Рейтинг популярности событий по количеству.

In [23]:
```
#таблица популярности событий
rating_event = game_actions['event'].value_counts().reset_index()
rating_event['ratio_%'] = round(rating_event['event'] / rating_event['event'].sum() * 100, 2)
display(rating_event)

#круговая диграмма популярности событий
fig = px.pie(rating_event, values='event', names = 'index', title='Рейтинг популярности событий по количеству.'
fig.show()
```

	index	event	ratio_%
0	building	103637	93.10
1	finished_stage_1	5817	5.23
2	project	1866	1.68

Мы наблюдаем, что 93% всех событий занимают постройки. 5% все событий это переход на новый уровень, а оставшиеся 1.6% это реализация проекта.

Рейтинг популярности строящихся объектов.

In [24]:
```
#таблица популярных строящихся объектов
rating_building = game_actions['building_type'].value_counts().reset_index()
rating_building['ratio_%'] = round(rating_building['building_type'] / rating_building['building_type'].sum() *
display(rating_building)

#круговая диграмма популярных построек
fig = px.pie(rating_building, values='building_type', names = 'index', title='Рейтинг популярности построек по
fig.show()
```

	index	building_type	ratio_%
0	spaceport	49453	47.72
1	assembly_shop	41476	40.02
2	research_center	12708	12.26

Мы наблюдаем, что почти половину всех построек - 48% занимают космопорты. 40% построек занимают сборочные цехи, а оставшиеся 12% это исследовательские центры

Общий рейтинг рекламных источников.

Учитывая все предыдущие рейтинги и найденную закономерность, в виде совпадения рейтингов по всем расчетам - составим общий рейтинг рекламных источников:

1) Яндекс

2) Ютуб

3) Инстаграм

4) Фэйсбук

Яндекс: один из самых дорогих источников по стоимости кликов, который является фаворитом по количеству приведенных пользователей - 36%, поэтому на него выделяется наибольшее количество средств - 40% всего бюджета.

Инстаграм: так же является одним из самых дорогих по стоимости кликов, который привел 25% всех пользователей и на которой потратили 26% всех средств.

Фэйсбук: немного отстает от инстаграма, чуть дешевле стоят клики, привел 20% пользователей и имеет общие затраты 22%

Ютуб: своего рода аутсайдер. Стоимость на клики в два раза меньше, чем стоимость кликов у его конкурентов, однако привел он 19% польователей. Данный показатель не сильно отстает от инстаграма и фэйсбука. При этом общие траты - 11%, что в 4 раза меньше чем траты на Яндекс и в 2 раза меньше, чем траты на другие источники.

Учитывая расчеты и анализ этих источников, я бы обратил особое внимание на Яндекс (стоит дорого, приводит много, затраты большие) и на Ютуб (стоит дешево, приводит достаточно много в сравнении с другими источниками и очень низкие затраты)

Анализ влияния событий на совершение целевого события.

Разбивка пользователей на категории в зависимости от их способа перехода на второй уровень и их сравнение.

In [25]:
```
#создаем 3 категории пользователей - воины, строители и те, кто не достиг 2 уровня
category_builders = game_actions.query('event == "project"')['user_id'].unique()
category_fighters =  game_actions.query('event == "finished_stage_1" and user_id not in @category_builders')['u
category_first_level = game_actions.query('user_id not in @category_builders and user_id not in @category_fight

user_info = user_source

def category(user):
```

```
    '''
    В данной функции мы присваиваем каждому user_id категорию
    в зависимости от его действий в игре
    '''
    if user in category_builders:
        return 'builder'
    elif user in category_fighters:
        return 'fighter'
    elif user in category_first_level:
        return 'first_level'

user_info['category'] = user_info['user_id'].apply(category)

first_level = user_info

#таблица пользователей по категориям
category_user = user_info.groupby('category')['user_id'].count().reset_index()
category_user['ratio_%'] = round(category_user['user_id'] / category_user['user_id'].sum() * 100 , 2)

display(category_user)

#круговая диграмма пользователей по категориям
fig = px.pie(category_user, values='user_id', names = 'category', title='Разбивка пользователей по категориям.'
fig.show()

#таблица пользователей второго уровня по пользователям
category_second_level = user_info.query('category == "builder" or category == "fighter"').groupby('category')['
category_second_level['ratio_%'] = round(category_second_level['user_id'] / category_second_level['user_id'].su

display(category_second_level)

#круговая диаграмма пользователей второго уровня по пользователям
fig = px.pie(category_second_level, values='user_id', names = 'category', title='Разбивка пользователей второго
fig.show()
```

	category	user_id	ratio_%
0	builder	1866	13.74
1	fighter	3951	29.10
2	first_level	7759	57.15

	category	user_id	ratio_%
0	builder	1866	32.08
1	fighter	3951	67.92

Мы можем наблюдать, что лишь 43% пользователей доходят до второго уровня. Из них 68% это воины, которые либо сразу перешли на новый уровень путем сражения или имели постройки, но так и не завершили проект. Оставшиеся 32% - строители, которые закончили проект.

Расчет времени, которое тратят пользователи на достижение второго уровня с момента регистрации.

In [26]:
```
#оставляем только пользователей, перешедших на второй уровень
user_info = user_info.query('category == "builder" or category == "fighter"').merge(game_actions, on = 'user_id
```

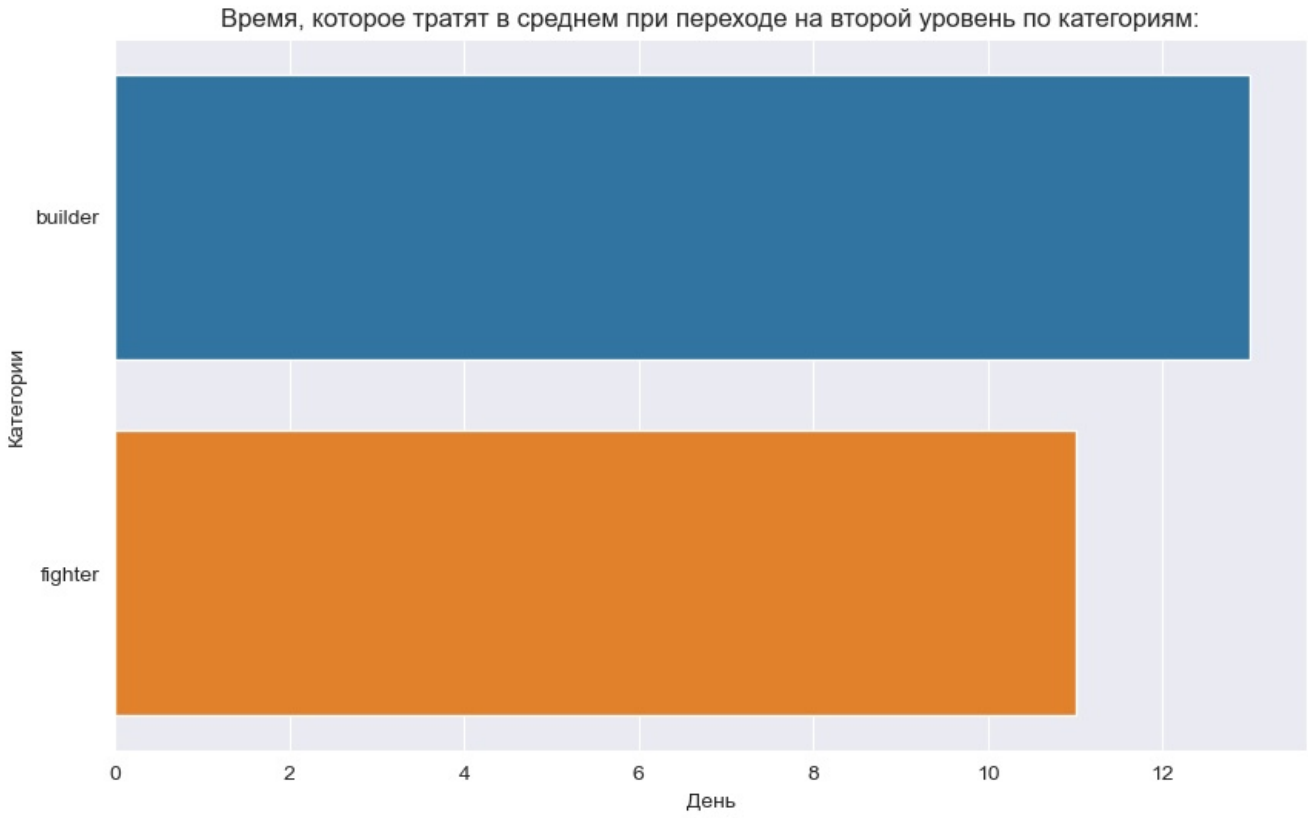
In [27]:
```
#находим первый и последний день активности пользователя
min_date = user_info.groupby('user_id')['event_datetime'].min().reset_index()
max_date = user_info.groupby('user_id')['event_datetime'].max().reset_index()
table_date = min_date.merge(max_date, on = 'user_id')

#путем вычета дат находим количество дней, которые тратят пользователи на достижение второго уровня
table_date['days'] = table_date['event_datetime_y'] - table_date['event_datetime_x']

table_date = table_date[['user_id', 'days']]

user_info = user_info.merge(table_date, on = 'user_id')

#переводим тип данных из datetime в int
user_info['days'] = user_info['days'].dt.days
```

In [28]:
```
#таблица времени, которое в среднем тратят пользователи на достижение второго уровня по категориям
time_category = user_info.groupby('category')['days'].median().reset_index()

display('Fighter', user_info.query('category == "fighter"')['days'].describe())
display('Builder', user_info.query('category == "builder"')['days'].describe())
display(time_category)

#график медианного времени, которое в среднем тратят пользователи разных категорий на достижение второго уровня
plt.figure(figsize = (10,6))
sns.set_style("darkgrid")
sns.barplot(data = time_category, x = 'days', y = 'category')
plt.title('Время, которое тратят в среднем при переходе на второй уровень по категориям:')
plt.xlabel('День')
plt.ylabel('Категории')
plt.show()
```

```
'Fighter'
count    34149.000000
mean        11.707312
std          4.002223
min          0.000000
25%          9.000000
50%         11.000000
75%         14.000000
max         31.000000
Name: days, dtype: float64
```

```
'Builder'
count    22470.000000
mean        13.543525
std          3.534681
min          5.000000
25%         11.000000
50%         13.000000
75%         16.000000
max         29.000000
Name: days, dtype: float64
```

	category	days
0	builder	13.0
1	fighter	11.0

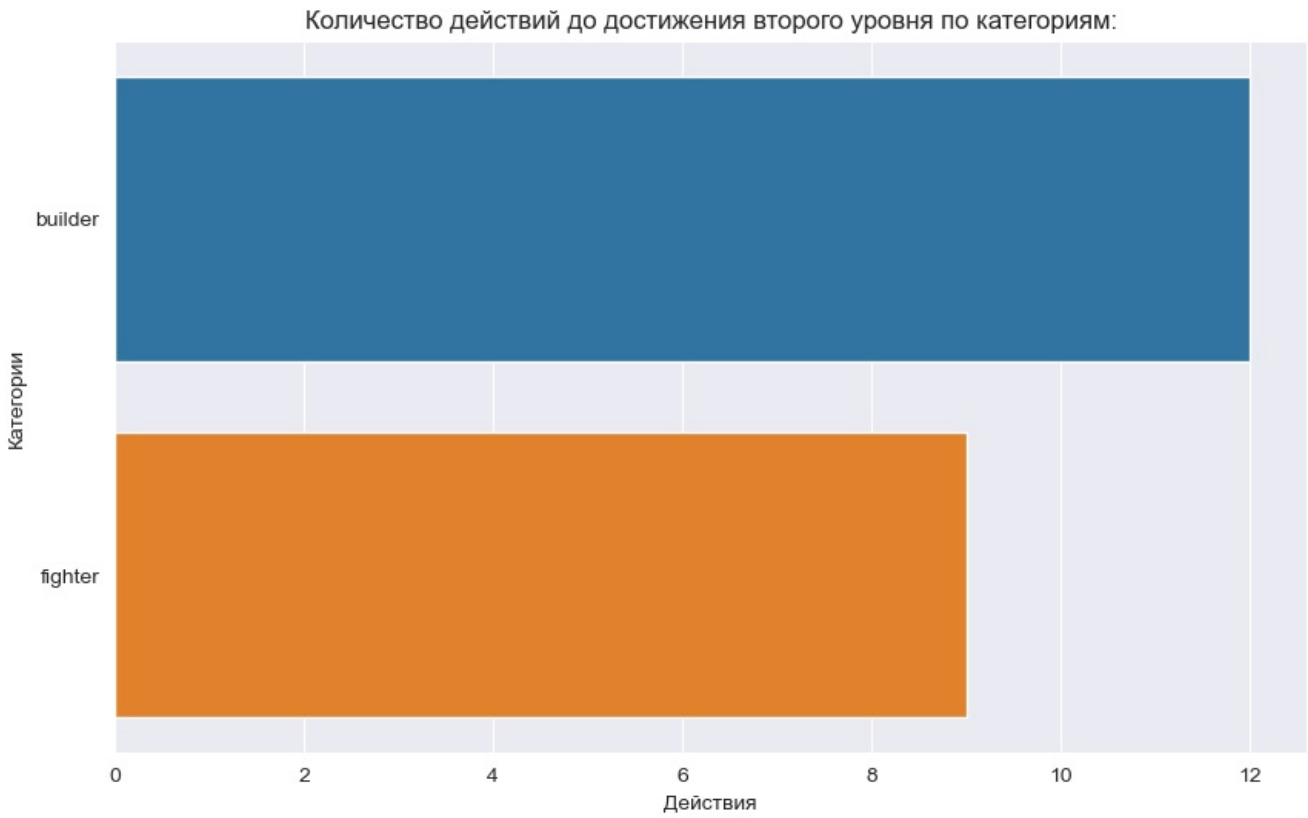


Как мы можем наблюдать, в среднем строители тратят 13 дней на переход к второму уровню. Воины тратят на два дня меньше - 11 дней. Учитывая, что пользователи могут сидеть на первом уровне сколько угодно и строить новый объекты без ограничений, мы можем предположить, что воины все таки стараются скорее перейти на новый уровень. Поскольку они воины - им не так нравится заниматься строительством и переход на новый уровень занимает в среднем 11 дней, что на два дня меньше, чем у любителей новых построек. Заметим, что воинов в два раза больше чем строителей.

Анализ действий пользователей до достижения finished_stage_1.

In [29]:
```
events_to_second_level = user_info.groupby(['user_id','category'])['event'].count().reset_index()
#таблица количества событий в среднем по категориям
events_categories = events_to_second_level.groupby('category')['event'].median().reset_index()

#таблица объектов, которые строят строители
buildings_builder = user_info.query('category == "builder"')['building_type'].value_counts().reset_index()
buildings_builder['ratio_%'] = round(buildings_builder['building_type'] / buildings_builder['building_type'].su

#таблица объектов, которые строят воины
buildings_fighter = user_info.query('category == "fighter"')['building_type'].value_counts().reset_index()
buildings_fighter['ratio_%'] = round(buildings_fighter['building_type'] / buildings_fighter['building_type'].su

display(events_categories)

#график количества действий по категориям
plt.figure(figsize = (10,6))
sns.set_style("darkgrid")
sns.barplot(data = events_categories, x = 'event', y = 'category')
plt.title('Количество действий до достижения второго уровня по категориям:')
plt.xlabel('Действия')
plt.ylabel('Категории')
plt.show()
```

	category	event
0	builder	12.0
1	fighter	9.0



Как мы видим наше предположение из пункта 4.2 имеет место быть. Категория воинов действительно занимается строительством меньше, чем категория строителей. В среднем воины строят по 9 объектов, в то время, как категория строителей строит по 12 объектов в среднем.

In [30]:
```
display(buildings_builder)

#круговая диаграмма объектов, которые строят строители
fig = px.bar(buildings_builder, x='index', y='building_type', text = 'ratio_%', title = 'Объекты, которые строи
fig.update_layout(xaxis_title="Объекты",yaxis_title="Количество")
fig.show()

display(buildings_fighter)

#круговая диаграмма объектов, которые строят воины
fig = px.bar(buildings_fighter, x='index', y='building_type', text = 'ratio_%', title = 'Объекты, которые строи
fig.update_layout(xaxis_title="Объекты",yaxis_title="Количество")
fig.show()
```

	index	building_type	ratio_%
0	spaceport	8759	46.74
1	assembly_shop	6675	35.62
2	research_center	3304	17.63

	index	building_type	ratio_%
0	spaceport	14398	47.68
1	assembly_shop	12386	41.02
2	research_center	3414	11.31

Что касаемо объектов, которые строят пользователи на пути к второму уровню, то показатели не сильно отличаются. В обеих категориях фаворитом среди построек является космопорт, который занимает 47-48%. На втором месте сборочный цех, имеющий показатель 36 и 41%. Реже всего строят исследовательський центр, который занимает 18 и 11% соответственно.

In [49]:
```
import plotly.express as px
buildings_builder_rev = buildings_builder
buildings_builder_rev['strategy'] = 'builder'
buildings_fighter['strategy'] = 'fighter'

df_rev = pd.concat([buildings_builder, buildings_fighter])
df_rev

fig = px.bar(df_rev, x="strategy", y="building_type", color="index")
```

```
fig.update_layout(title = 'Соотношение объектов, которые строят игроки', xaxis_title="Стратегия",yaxis_title="T
fig.show()
```

Расчет пользователей, которые останавливаются, так и не совершив переход на новый уровень.

In [32]:
```
#датафрейм с категорией людей, которые не достигли второго уровня
first_level = first_level.query('category == "first_level"').merge(game_actions, on = 'user_id')
```

In [33]:
```
min_date = first_level.groupby('user_id')['event_datetime'].min().reset_index()
max_date = first_level.groupby('user_id')['event_datetime'].max().reset_index()
table_date = min_date.merge(max_date, on = 'user_id')

#путем вычета дат находим количество дней, которые пользователи проводят в игре
table_date['days'] = table_date['event_datetime_y'] - table_date['event_datetime_x']

table_date = table_date[['user_id', 'days']]

first_level = first_level.merge(table_date, on = 'user_id')
first_level['days'] = first_level['days'].dt.days
```

In [34]:
```
#круговая диаграмма пользователей по категориям
fig = px.pie(category_user, values='user_id', names = 'category', title='Разбивка пользователей по категориям.'
fig.show()

#таблица объектов, которые строят пользователи, недошедшие до второго уровня
first_level_buildings = first_level['building_type'].value_counts().reset_index()
first_level_buildings['ratio_%'] = round(first_level_buildings['building_type'] / first_level_buildings['buildi
display(first_level_buildings)

#круговая диаграмма объектов, которые строят игроки первого уровня
fig = px.bar(first_level_buildings, x='index', y='building_type', text = 'ratio_%', title = 'Объекты, которые с
fig.update_layout(xaxis_title="Объекты",yaxis_title="Количество")
fig.show()
```

	index	building_type	ratio_%
0	spaceport	26296	48.07
1	assembly_shop	22415	40.98
2	research_center	5990	10.95

Как мы можем наблюдать, 58% всех пользователей игры не дошли до второго уровня. При этом показатели по строительству объектов не отличаются от тех же показателей у строителей и воинов.

Рейтинг источников по категориям игроков:

In [35]:
```
#создаем отдельный фрейм с категорией строителей
builders = user_info.query('category == "builder"')

#таблица распределения источников среди строителей
builders_source = builders['source'].value_counts().reset_index()
builders_source['ratio_%'] = round(builders_source['source'] / builders_source['source'].sum() * 100, 2)
```

```
display(builders_source)

#круговая диаграмма распределения источников среди строителлей
fig = px.bar(builders_source, x='index', y='source', text = 'ratio_%', title = 'Распределение источников по кат
fig.update_layout(xaxis_title="Источники",yaxis_title="Количество")
fig.show()
```

	index	source	ratio_%
0	yandex_direct	7435	33.09
1	instagram_new_adverts	5759	25.63
2	facebook_ads	4737	21.08
3	youtube_channel_reklama	4539	20.20

In [36]:
```
#создаем отдельный фрейм с категорией воинов
fighters = user_info.query('category == "fighter"')

#создаем табицу распределения источников среди воинов
fighters_source = fighters['source'].value_counts().reset_index()
fighters_source['ratio_%'] = round(fighters_source['source'] / fighters_source['source'].sum() * 100, 2)

display(fighters_source)

#круговая диаграмма распределения источников среди воинов
fig = px.bar(fighters_source, x='index', y='source', text = 'ratio_%', title = 'Распределение источников по кат
fig.update_layout(xaxis_title="Источники",yaxis_title="Количество")
fig.show()
```

	index	source	ratio_%
0	yandex_direct	12274	35.94
1	instagram_new_adverts	8563	25.08
2	youtube_channel_reklama	6758	19.79
3	facebook_ads	6554	19.19

In [37]:
```
#таблица распредлеения источников среди первого уровня
first_level_source = first_level['source'].value_counts().reset_index()
first_level_source['ratio_%'] = round(first_level_source['source'] / first_level_source['source'].sum() * 100,

display(first_level_source)

#круговая диаграмма распределения источников среди первого уровня
fig = px.bar(first_level_source, x='index', y='source', text = 'ratio_%', title = 'Распределение источников сре
fig.update_layout(xaxis_title="Источники",yaxis_title="Количество")
fig.show()
```

	index	source	ratio_%
0	yandex_direct	19519	35.68
1	instagram_new_adverts	13320	24.35
2	facebook_ads	11387	20.82
3	youtube_channel_reklama	10475	19.15

В целом можно заметить, что распределение по источникам не особо отличается от категории к категории.

Рейтинг источников по категориям игроков:

1) Яндекс

2) Инстаграм

3) Фейсбук

4) Ютуб

Проверка статистических гипотез

Гипотеза о различии времени прохождения уровня между пользователями, которые заканчивают уровень через реализацию проекта, и пользователями, которые заканчивают уровень победой над другим игроком.

Необходимо проверить гипотезу о различии времени прохождения уровня между пользователями, которые заканчивают уровень через реализацию проекта, и пользователями, которые заканчивают уровень победой над другим игроком.

Составим нулевую и альтернативную гипотезу о равенстве/неравенстве средних двух генеральных совокупностей.

H0: Время прохождение уровня между пользователями разных категорий не отличается

H1: Время прохождение уровня между пользователями разных категорий различается

In [38]:
```
#создаем фрейм с пользователями категории строителями
group_project = user_info.query('category == "builder"')
group_project = group_project[['user_id','days']]
group_project = group_project.drop_duplicates()['days']

#создаем фрейм с пользователями категории воины
group_fight = user_info.query('category == "fighter"')
group_fight = group_fight[['user_id','days']]
group_fight = group_fight.drop_duplicates()['days']

display('Количество пользователей строителей:',group_project.count())
display('Количество пользователей воинов:',group_fight.count())
```

```
'Количество пользователей строителей:'
1866
'Количество пользователей воинов:'
3951
```

In [39]:
```
#проводим статистический ttest
results = st.ttest_ind(group_project, group_fight, equal_var = False)

#объявляем пороговое значение 5%
alpha = 0.05

print('pvalue:',results.pvalue)

if results.pvalue > alpha:
    print('Оставляем нулевую гипотезу')
else:
    print('Отказываемся от нулевой гипотезы')
```

```
pvalue: 6.5704667556440105e-105
Отказываемся от нулевой гипотезы
```

Для проверки гипотезы использовали ttest с параметром equal_var = False, поскольку совокупности разного размера. Пороговое значение - 5%. Как мы видим, статистический тест лишь подтвержает наши исследования выше, каждая категория пользователей проходит уровень разное время.

Гипотеза о различии трат за клики на Инстаграм и Фэйсбук

Проверим и гипотезу о различии трат на такие источники, как Инстаграм и Фэйсбук. Ранее мы обнаружили, что эти источники конкурируют друг с другом.

Составим нулевую и альтернативную гипотезу о равенстве/неравенстве средних двух генеральных совокупностей.

H0: Траты на источники за весь период не отличаются.

H1: Траты на источники за весь период различаются.

In [40]:
```
#траты на инстаграм по дням
instagram = rating_cost['instagram_new_adverts']

#траты на фэйсбук по дням
```

```
facebook = rating_cost['facebook_ads']
```

In [41]:
```
#проводим ttest
results = st.ttest_ind(instagram , facebook, equal_var = True)

#обьявляем пороговое значение 5%
alpha = 0.05

print('pvalue:',results.pvalue)

if results.pvalue > alpha:
    print('Оставляем нулевую гипотезу')
else:
    print('Отказываемся от нулевой гипотезы')
```

```
pvalue: 0.9871252038135099
Оставляем нулевую гипотезу
```

Для проверки гипотезы использовали ttest с параметром equal_var = True, поскольку совокупности одного размера. Пороговое значение - 5%. Статистический тест сообщает, что статистической разницы в тратах на источники нет.

Подведение итогов и презентация результатов

Проведен исследовательский анализ данных, на основе результатов мы можем дать некоторые рекомендации, а также сделать выводы:

Общий рейтинг рекламных источников: 1) Яндекс

2) Инстаграм

3) Фэйсбук

4) Ютуб

Выводы:

- Лучшим рекламным источником по количеству приводимых пользователей - Яндекс, на него прихожится 35% новых пользователей. Однако важно учитывать, что это самый дорогой источник.
- Источники Фэйсбук и Инстаграм показали не самые лучшие результаты, несмотря на то, что стоимость кликов на эти исчтоники немного меньше чем у Яндекса - они провели относительно небольшое количество людей за ту стоимость, которую просят.
- Рекламный источник Ютуб является лучшим по соотношению цена/качество. Данный источник привел как и Фэйсбук 20% пользователей, однако средняя цена за клики в 2 раза меньше чем у его конкурентов при этом траты на Ютуб всего 10% от всех трат.
- Почти половину всех построек - 48% занимают космопорты. 40% построек занимают сборочные цехи, а оставшиеся 12% это исследовательские центры.
- 57% пользователей не доходят до второго уровня. Оставшиеся 43% пользователей доходят до второго уровня. Из них 68% это воины, которые либо сразу перешли на новый уровень путем сражения или имели постройки, но так и не завершили проект. Оставшиеся 32% - строители, которые закончили проект.
- В среднем строители тратят 13 дней на переход к второму уровню, а воины тратят 11 дней.
- В среднем воины строят по 9 объектов, строители строят по 12 объектов

Модель монетизации:

Изначальная идея показа рекламы во время строительства достаточно здравая, исходя из полученных результатов, однако ее стоит расширить.

- В игре 68% воинов. Можно добавить показ рекламы перед битвой, это увеличит прибыль. Тогда в среднем воины будут смотреть по 10 роликов, а строители по 12 роликов.
- Рекомендую дать возможность игрокам участвовать в битве в любой момент игры. Исходя из предложения выше, игроки, будучи уверенными в своих силах будут чаще сражаться в битвах с желанием перейти на новый уровень, соответственно он будет смотреть рекламу перед сражением. Желание людей поскорее развиться лишь сыграет компании на руку.
- Исследовательские центры занимают лишь 12% всех объектов. Нужно усилить этот объект, чтобы он имел большее значение в развитии персонажа. Так пользователь будет более замотивирован его строительством, а соответственно посмотрит рекламу перед началом постройки.
- Рекомендую добавить большее количество объектов и соответственно большее количество связей между

объектами(ресурсы, энергия и т.д). За пример можно взять игру Clash of Clans от SuperCell. Больше объектов -> сложнее игра -> больше просмотров рекламы -> большая вовлеченность игроков и увеличение времени на прохождение уровня.

- С расширением игры и увеличением объектов можно добавить донат, который дает мгновенное развитие. В последствии добавить временный ивент в игре, в котором люди будут соревноваться и желая занять призовые места - будут донатить.
- Чем больше рекламы, тем больше она надоедает игроку и тем быстрее он бросает игру. Можно добавить платную функцию отключения рекламы.